Felicia Ionescu

PROGRAMAREA APLICAŢIILOR GRAFICE

Editura PRINTECH
Bucureşti

Descrierea CIP a Bibliotecii Naţionale

IONESCU, FELICIA

Programarea aplicaţiilor grafice.

Felicia Ionescu, Bucureşti: PRINTECH, 1999

203 p. :il. ; cm.
ISBN 973-9475-93-0

1.Programare
004.42

Referenţi ştiinţifici:
Prof. Dr. Ing. R. Strungaru
Ş.l. Dr. Ing. O. Grigore

Editura PRINTECH
Splaiul Independenţei 313, Bucureşti, Sector 6
Tel: 092 355 664

Director Editură:
Ecaterina Milică

Consilier Editorial:
Mihaela Toma

Toate drepturile asupra acestei ediţii aparţin autorului. Reproducerea parţială sau totală a textului sau a ilustraţiilor din această carte este posibilă numai cu acordul scris al autorului.

Cuprins

1 Introducere în grafica pe calculator 1

1.1 Aplicaţii ale graficii pe calculator 1
1.2 Sisteme grafice 3
1.3 Sisteme de referinţă tridimensionale 4

2 Modelarea obiectelor 7

2.1 Modelarea poligonală a obiectelor 8
2.1.1 Reprezentarea poligoanelor 8
2.1.2 Reprezentarea poliedrelor 11
2.1.3 Implementarea modelului poligonal 13
2.1.4 Generarea modelului din descrierea matematică 16
2.1.5 Generarea modelului prin baleiere spaţială 17
2.1.6 Generarea modelului pornind de la o mulţime de puncte care aparţin suprafeţei de frontieră a obiectului 18
2.1.7 Redarea imaginii obiectelor poligonale 19
2.2 Modelarea obiectelor prin reţele de petice 8
2.3 Modelarea prin compunerea obiectelor 21
2.4 Modelarea prin divizare spaţială 22

3 Transformări geometrice 25

3.1 Transformări geometrice în spaţiu 25
3.1.1 Transformări geometrice primitive 25
3.1.2 Sistemul de coordonate omogene 28
3.1.3 Compunerea transformărilor geometrice 31
3.1.4 Transformări inverse 35
3.1.5 Transformarea sistemelor de referinţă 37
3.1.5.1 Rotaţia faţă de o axă paralelă cu una din axele sistemului de referinţă 40
3.1.5.2 Scalarea faţă de un punct oarecare în spaţiu 42
3.1.5.3 Rotaţia faţă de o dreaptă oarecare în spaţiu 42
3.1.5.4 Transformarea de modelare 44
3.1.6 Alte transformări geometrice în spaţiu 46
3.2 Transformări geometrice în plan 47

4 Sisteme de vizualizare 49

4.1 Transformarea de observare 51
4.2 Transformarea de proiecţie 56

4.2.1 Proiecţia paralelă 57
4.2.2 Proiecţia perspectivă 57
4.2.3 Sistemul de referinţă normalizat 59
4.3 Sistemul de vizualizare standard 66
4.3.1 Definirea sistemului de referinţă de observare 67
4.3.2 Definirea transformării de normalizare 67
4.4 Sistemul de referinţă ecran 3D 72
4.5 Decuparea obiectelor 74
4.5.1 Decuparea în plan 74
4.5.2 Decuparea suprafeţelor relativ la volumul de vizualizare 79
4.5.3 Eliminarea obiectelor aflate în exteriorul volumului de vizualizare 82
4.5.4 Volumul de delimitare 83
4.5.5 Detecţia coliziunii 84

5 Transformarea de rastru 85

5.1 Reprezentarea culorilor în sistemele grafice 86
5.1.1 Modelul RGB 86
5.1.2 Modelul HSV 88
5.2 Generarea segmentelor de dreaptă 89
5.3 Generarea poligoanelor 91
5.4 Eliminarea suprafeţelor ascunse 93
5.4.1 Compararea adâncimilor 94
5.4.2 Eliminarea suprafeţelor ascunse în spaţiul obiect 95
5.4.3 Eliminarea suprafeţelor ascunse în spaţiul imagine: algoritmul Z-buffer 96
5.4.4 Eliminarea suprafeţelor orientate invers 99
5.5 Anti-aliasing 102
5.6 Texturarea 103
5.6.1 Aplicaţia texturilor 104
5.6.2 Filtrarea texturilor 106

6 Biblioteca grafică OpenGL 109

6.1 Dezvoltarea programelor grafice folosind utilitarul GLUT 111
6.1.1 Funcţii de control ale ferestrei de afişare 111
6.1.2 Funcţii callback 112
6.1.3 Funcţii de execuţie 113
6.1.4 Generarea obiectelor tridimensionale 113
6.2 Caracteristicile bioblioteci OpenGL 114
6.2.1 Poarta de afişare OpenGL 114
6.2.2 Bufferul de cadru 115
6.2.3 Operaţiile de bază OpenGL 117
6.2.4 Primitive geometrice 120
6.2.5 Reprezentarea culorilor în OpenGL 123

6.3 Sistemul de vizualizare OpenGL 125
6.3.1 Sistemele de referinţă 125
6.3.2 Transformări geometrice 127
6.4 Decuparea suprafeţelor şi eliminarea obiectelor aflate în exteriorul volumului de vizualizare 141
6.4.1 Eliminarea suprafeţelor ascunse 142
6.4.2 Selecţia suprafeţelor în funcţie de orientare 142
6.5 Liste de display OpenGL 143
6.6 Texturarea în OpenGL 145
6.6.1 Definirea texturilor 145
6.6.2 Filtrarea texturilor 152

7 Modele de reflexie şi iluminare 157

7.1 Consideraţii teoretice asupra reflexiei luminii 157
7.2 Modelul Phong de reflexie a luminii 160
7.3 Modele de umbrire 162
7.3.1 Modelul de umbrire constantă 162
7.3.2 Modelul de umbrire Gouraud 162
7.3.3 Modelul de umbrire Phong 166
7.4 Funcţiile OpenGL de calcul a iluminării 167
7.4.1 Definirea surselor de lumină 167
7.4.2 Definirea proprietăţilor materialelor 169
7.4.3 Normalele în vârfurile primitivelor geometrice 174
7.4.4 Controlul poziţiei şi al direcţiei surselor de lumină 176
7.4.5 Combinarea culorilor 180
7.4.5.1 Transparenţa 181
7.4.5.2 Anti-aliasing 184

8 Crearea şi vizualizarea scenelor virtuale 187

8.1 Reprezentarea scenelor virtuale 189
8.2 Limbajul VRML 192
8.2.1 Construirea formelor geometrice 195
8.2.2 Noduri de grupare 196
8.2.3 Animaţia în VRML 201

Bibliografie 203

Prefaţă

Domeniile de aplicaţie ale graficii pe calculator s-au extins o dată cu creşterea puterii de calcul a calculatoarelor şi cu ieftinirea adaptoarelor şi a acceleratoarelor grafice: realizarea interfeţelor utilizator dezvoltate în numeroase programe utilitare şi medii de programare, proiectarea asistată de calculator (CAD – *Computer Aided Design*), prezentările grafice interactive, vizualizarea datelor ştiinţifice, tehnologia multimedia, realitatea virtuală. În acelaşi timp, apariţia a numeroase biblioteci, limbaje şi programe utilitare (toolkits) care propun diferite modalităţi de abordare a aplicaţiilor grafice face şi mai dificilă sarcina programatorului de a selecta cea mai adecvată tehnică de soluţionare a problemei dorite.

În această lucrare se pune accentul pe conexiunea dintre aspectele teoretice ale graficii şi implementarea programelor de generare a imaginii obiectelor şi scenelor virtuale tridimensionale. Lucrarea se bazează pe materialul de curs *Grafica pe Calculator*, predat la anul IV, specializarea Ingineria Informaţiei, secţia Electronică Aplicată şi Ingineria Informaţiei a Facultăţii de Electronică şi Telecomunicaţii. În primele capitole sunt prezentate operaţiile de bază în grafica pe calculator: modelarea obiectelor tridimensionale, transformări geometrice în spaţiu, sisteme de vizualizare, transformarea de rastru. În cea de-a doua parte a lucrării se reiau toate aceste aspecte ale generării imaginilor din perspectiva programării, folosind biblioteca grafică OpenGL, sistemul de dezvoltare GLUT şi limbajul de modelare în realitatea virtuală VRML. De asemenea, sunt prezentate aspectele avansate ale generării imaginilor tridimensionale (anti-aliasing, umbrire, texturare), atât din punct de vedere teoretic, cât şi al abordării în programare.

Unul din scopuri principale ale cărţii este de a oferi soluţii de implementare a aplicaţiilor grafice prin prezentarea metodelor actuale şi accesibile de rezolvare a fiecărei etape a procesului de generare a imaginilor. În acest sens, aproape toate imaginile ilustrative ale diferitelor tehnici au fost produse de autoare prin programe grafice descrise în text.

Octombrie, 1999 Autoarea

1

Introducere în grafica pe calculator

În standardul ISO (International Standardization Office) se defineşte grafica pe calculator (*computer graphics*) ca "metode şi tehnici de conversie a datelor către şi de la un dispozitiv grafic (display) prin intermediul calculatorului". În plus, se mai menţionează că aceste metode reprezintă modul cel mai puternic de comunicare între om şi calculator.

Informaţia prezentată vizual este percepută de om în modul cel mai natural. Structuri complexe şi relaţiile între ele pot fi percepute într-un timp mai scurt, într-un număr mai mare şi cu mai puţine erori în prezentarea vizuală decât în oricare alt mod de prezentare.

Noţiunea de *grafică pe calculator* este asimilată în cele mai multe referinţe din domeniu cu aceea de grafică generativă (sau sinteză de imagine), având ca direcţie de conversie şi transformare a datelor aceea de a crea pe display imaginea unor obiecte a căror descriere este stocată în memoria calculatorului.

1.1 Aplicaţii ale graficii pe calculator

Domeniile de aplicaţie ale graficii pe calculator s-au extins o dată cu creşterea puterii de calcul a calculatoarelor şi cu ieftinirea adaptoarelor şi a acceleratoarelor grafice. În continuare sunt prezentate câteva din cele mai importante domenii de aplicaţie ale graficii pe calculator.

- Una din cele mai extinse utilizări ale graficii este în realizarea interfeţelor utilizator dezvoltate în numeroase programe utilitare, medii de dezvoltare şi programare, sisteme de operare, baze de date. Intefeţele grafice sunt mult mai "prietenoase" decât interfeţele prin comenzi date în linia de comandă şi permit instruirea rapidă a utilizatorilor neexeperimentaţi în folosirea unor astfel de aplicaţii.

- O altă ramură importantă de aplicare a graficii pe calculator o constitue proiectarea asistată de calculator (CAD – *Computer Aided Design*), folosită în electronică, mecanică, aeronautică, arhitectură, etc. Domeniul de proiectare asistată CAD a reprezentat multă vreme cea mai importantă aplicaţie a graficii pe calculator şi a prilejuit dezvoltarea a numeroase tehnici de generare şi vizualizare a modelelor tridimensionale.
- Prezentările grafice interactive (în administraţie, tehnologie, statistică, management, planificare, comanda şi conducerea proceselor industriale) generează o gamă variată de reprezentări sugestive ale datelor numerice. Astfel de prezentări permit analiza sistemelor mult mai sugestiv decât datele prezentate în tabele.
- În tehnologia multimedia prezentarea de imagini grafice sintetizate, texte, imagini video şi sunet sunt integrate într-o singură aplicaţie (de instruire interactivă, teleconferinţe, etc). Conexiunea dintre generarea de imagini tridimensionale şi prezentarea de imagini video animate reprezintă cel mai important aspect al soluţiei multimedia.
- *Realitatea virtuală* este tehnica prin care se obţine imersiunea (totală sau parţială) a observatorului într-un mediu tridimensional generat interactiv de calculator.

Realitatea Virtuală a cunoscut în ultimii ani o imensă dezvoltare şi publicitate. În reviste, magazine, la televiziune, s-a prezentat această "nouă şi fantastică tehnologie" din cele mai variate puncte de vedere. Cuvântul "virtual" este folosit în mod obişnuit în domeniul calculatoarelor pentru a desemna o entitate care simulează o altă entitate. De exemplu, termenul "memorie virtuală" se referă la simularea memoriei principale prin memoria hard-discului. Cuvântul "realitate" se referă la mediul perceput de om prin intermediul simţurilor. Deoarece "realitatea" este percepută prin intermediul simţurilor, este posibilă "alterarea" acesteia prin modificarea datelor percepute de unul sau mai multe simţuri. De aceea, *Realitatea Virtuală* se referă la modalitatea prin care calculatorul modifică modul în care o persoană percepe realitatea, prin simularea unei alte realităţi. Această realitate, sau mediu, simulat de calculator este numită *Realitate Virtuală*.

Dintre cele cinci simţuri care sunt folosite pentru percepţia realităţii, nu toate sunt la fel de importante în crearea unui mediu virtual. Simţul gustului şi al mirosului au acţiuni limitate în perceperea realităţii (cu excepţia servirii mesei!) şi puţine cercetări au fost efectuate pentru folosirea lor în medii virtuale. Simţul tactil este mult mai util, mai ales atunci când se manipulează obiecte în mediul virtual. Greutatea, temperatura, duritatea, rezistenţa la effort, toate aceste informaţii se obţin prin simţul tactil. Din acest motiv, cercetări importante au fost făcute pentru a simula "atingerea" obiectelor virtuale şi, în momentul de faţă, experimentele în mediul virtual permit generarea informaţiilor tactile. Cele mai importante simţuri folosite în realitatea virtuală sunt văzul şi auzul, deoarece cele mai multe informaţii despre mediul înconjurător se obţin prin intermediul ochilor şi al urechilor. Din acest motiv, cele mai multe cercetări în domeniul realităţii virtuale au fost făcute în domeniul generării imaginii şi a sunetelor. Dintre aplicaţiile cele mai importante ale realităţii virtuale se pot enumera:

- Simulatoare de antrenament, în special simulatoare de zbor, în care se pot exersa manevre dificile, fără a pune în pericol viaţa pilotului sau securitatea aparatului de zbor.
- Proiectare în diferite domenii de activitate (construcţii, arhitectură). Proiectantul are posibilitatea "să vadă" rezultatele proiectului sub forma imaginii acestuia în timp real, să observe detaliile împreună cu alte persoane interesate, şi să ia decizii de modificare înainte de construirea prototipului.
- Vizualizarea ştiinţifică, prin care se obţine imaginea diferitelor modele sau fenomene inaccesibile de fi observate altfel (structuri atomice, fluxuri de informaţie, etc).
- În domeniul medical, în special chirurgie, se pot efectua experimente "la rece" de învăţare a diferitelor proceduri fără riscul vieţii pacientului.
- Jocurile distractive şi filmele de animaţie sunt unele din cele mai cunoscute aplicaţii de realitate virtuală.

În aplicaţiile de realitate virtuală pot fi înglobate un număr diferit de componente care asigură un anumit grad de imersiune, adică de senzaţie a prezenţei participantului în mediul virtual: sistem de vizualizare, sistem de sunet, sistem de urmărire a poziţiei capului şi a mâinii, sistem generare a senzaţiei tactile şi a forţei de reacţie. În diferite aplicaţii de realitate virtuală, unele dintre aceste sisteme, cu excepţia sistemului de vizualizare, pot să lipsească.

Sistemul de vizualizare este componenta cea mai importantă a aplicaţiilor de realitate virtuală şi există unele categorii de aplicaţii de realitate virtuală (aplicaţii de realitate virtuală desktop) dezvoltate numai pe baza generării imaginii mediului virtual. În aplicaţiile de realitate virtuală desktop, imaginea vizuală a mediului virtual tridimensional este afişată pe monitorul unui calculator (în general PC). Participantul intercţionează cu mediul virtual prin dispozitive de intrare standard (tastatură, mouse, joystick). Aceste sisteme permit observarea mediului virtual printr-o "fereastră" (ecranul monitorului) şi de aceea se mai numesc sisteme WoW (*Window on the World*). Sunt cele mai simple şi mai ieftine sisteme de realitate virtuală, dar este de aşteptat ca astfel de sisteme să cunoască în viitor dezvoltări spectaculoase, datorită apariţiei unui mare număr de acceleratoare grafice care permit redarea în timp real a unor imagini realiste.

Crearea unui mediu virtual, în care se pot efectua diferite experimente, este un proces care necesită două componente importante, şi anume crearea modelului scenei virtuale şi vizualizarea scenei virtuale. Crearea modelului scenei virtuale (mai pe scurt, crearea scenei virtuale) este un proces off-line şi, de cele mai multe ori, de durată considerabilă, prin care se creează colecţia de modele ale obiectelor tridimensionale care constitue cea mai adecvată reprezentare a mediului virtual.

Vizualizarea scenelor virtuale este un proces on-line, care se desfăşoară în timp real, cu participarea uneia sau mai multor persoane, în care scena virtuală este explorată în mod interactiv, şi, în fiecare moment, imaginea scenei redată pe display depinde de condiţiile de explorare (poziţie de observare, acţiuni interactive, etc).

1.2 Sisteme grafice

O aplicaţie grafică, indiferent de domeniul căreia îi este destinată, se dezvoltă într-un sistem care prezintă anumite facilităţi grafice, sistem numit la modul general sistem grafic.

Un sistem grafic este un calculator care dispune de componente hardware şi software pentru crearea sau prelucrarea imaginilor. Componenta hardware cea mai importantă a unui sistem grafic este adaptorul (sau acceleratorul) grafic, prin intermediul căruia este comandat un dispozitiv de afişare (display) color. La aceasta se mai adaugă diferite dispozitive de intrare (tastatură, mouse, joystick, trackball, interfeţe specializate) care asigură interacţiunea utilizatorului cu programul de aplicaţie.

În programarea aplicaţiilor grafice intervin mai multe componente software, care asigură crearea imaginilor pe display (Fig. 1.1).

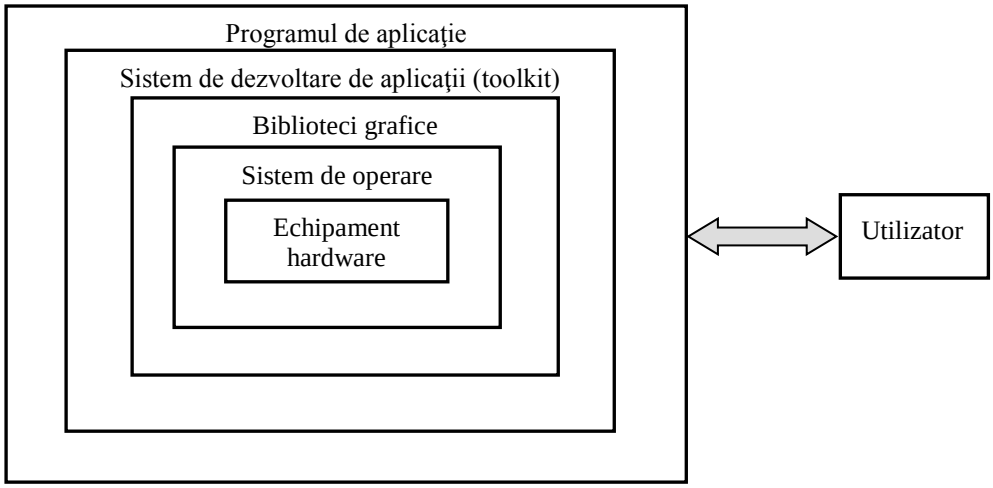


Fig. 1.1 Componentele necesare pentru crearea aplicaţiilor grafice.

Programul de aplicaţie grafică se dezvoltă pe baza unor sisteme de dezvoltare de aplicaţii (toolkit-uri) sau direct, prin utilizarea unor biblioteci grafice care asigură interfaţa cu echipamentul hardware prin intermediul driverelor sistemului de operare. Sistemele de dezvoltare sunt de cele mai multe ori orientate către aplicaţie şi prevăd un set de funcţii de nivel înalt care permit crearea unui anumit tip de aplicaţie. De exemplu, există toolkit-uri pentru generarea obiectelor şi a scenelor virtuale (3d Studio, Autocad, AC3d, Sense8, Designer Workbench, etc), toolkit-uri pentru redarea imaginii scenelor virtuale (Performer, browser Cosmo Player, etc).

Bibliotecile grafice sunt pachete de funcţii care asigură interfaţa programului de aplicaţie (creat direct sau prin intermediul unui toolkit care apelează funcţiile bibliotecii) cu echipamentele hardware ale sistemului grafic.

Bibliotecile grafice reprezintă nivelul de programare în care se încearcă introducerea portabilităţii programelor grafice, prin asigurarea unei interfeţe

independente de echipamentele hardware care să respecte anumite convenţii de reprezentare a entităţilor grafice descrise în standarde.

Primul standard grafic a fost standardul GKS (*Graphical Kernel System*) elaborat de ISO în 1985, care conţinea un set de funcţii grafice 2D independente de echipament. Acest standard a fost extins în anul 1988 la GKS 3D, care conţine funcţii 3D independente de echipament.

Un alt standard, PHIGS (*Programmer's Hierarchical Graphical System*) este un standard 3D care permite în plus organizarea ierarhică a modelelor obiectelor şi a scenelor virtuale.

Bibliotecile grafice cele mai generale sunt bibliotecile grafice care implementează un anumit standard în definirea funcţiilor de acces la echipamentele hardware. Cele mai cunoscute biblioteci grafice sunt OpenGL, DirectX, QuickDraw, care sunt implementate în numeroase sisteme grafice.

În momentul de faţă, situaţia cea mai obişnuită în crearea aplicaţiilor grafice este aceea în care se poate folosi un calculator care dispune de facilităţi grafice (adaptoare, acceleratoare, drivere, biblioteci) care permit programarea la nivel înalt, de cele mai multe ori complet independentă de dispozitivele hardware. Biblioteci grafice cum sunt OpenGL sau DirectX asigură accesul la funcţii grafice adaptate şi optimizate pentru dispozitivele grafice disponibile ale sistemului. De aceea, în lucrare s-a acordat o mai mică întindere descrierii unor funcţii existente în biblioteci (de exemplu funcţiile de generare a segmentelor de linie dreaptă, a cercurilor, etc.) şi accentul s-a îndreptat către prezentarea metodelor de bază de generare a imaginilor şi a modului în care acestea se pot aplica folosind limbaje şi biblioteci grafice de nivel înalt.

Programele grafice propuse în această carte se prezintă sub forma de cod C, C++ şi, uneori, pseudocod asemănător limbajului C. Biblioteca grafică OpenGL este folosită intens, datorită portabilităţii şi disponibilităţii acesteia în aproape toate calculatoarele, de la calculatoare PC până la staţii grafice puternice, sub un număr mare de sisteme de operare şi apelabilă din numeroase sisteme de dezvoltare (toolkit).

Scopul prezentei lucrări este acela de a da o descriere unificată a principalelor metode de generare a imaginilor tridimensionale împreună cu aspecte de implementare care să faciliteze sarcina programatorului de a aborda acest domeniu de aplicaţii diversificat şi în permanentă creştere. În acest sens, aproape toate imaginile ilustrative ale diferitelor tehnici au fost produse de autor prin programe grafice descrise în text.

1.3 Sisteme de referinţă tridimensionale

Pentru crearea aplicaţiilor grafice este necesar ca obiectele să fie poziţionate într-un sistem de referinţă tridimensional. Există mai multe posibilităţi de a specifica poziţia unei mulţimi de puncte (vârfuri) prin care este reprezentat un obiect în spaţiul tridimensional: coordonate cilindrice, coordonate sferice, coordonate Carteziene. Dintre aceste sisteme de referinţă, cel mai utilizat în aplicaţiile grafice este sistemul de coordonate Cartezian.

Sistemul de coordonate Cartezian în care sunt definite toate obiectele scenei virtuale se numeşte sistem de referinţă universal (*world coordinate system*- WCS).

Un sistem de coordonate Cartezian se defineşte prin originea O şi trei axe perpendiculare, Ox, Oy şi Oz, orientate după regula mâinii drepte sau după regula mâinii stângi. Diferenţa dintre cele două sisteme se poate urmări în Fig. 1.2. Într-un sistem orientat după regula mâinii drepte, dacă se roteşte mâna dreaptă în jurul axei z de la axa x pozitivă spre axa y pozitivă, orientarea degetului mare este în direcţia z pozitiv. Într-un sistem orientat după regula mâinii stângi, rotirea de la axa x pozitivă spre axa y pozitivă, cu orientarea degetului mare în direcţia z pozitiv se obţine folosind mâna stângă. Orientarea după regula mâinii drepte a sistemelor de coordonate corespunde convenţiei matematice standard.

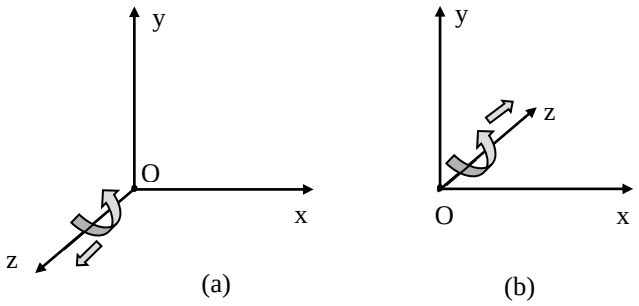


Fig.1.2 (a) Sistem de coordonate orientat după regula mâinii drepte şi (b) după regula mâinii stângi.

Diferite sisteme de grafică tridimensională sau de realitate virtuală folosesc convenţii diferite pentru definirea sistemelor de referinţă, ceea ce conduce la confuzii, dacă nu se precizează convenţia folosită. În acest text, pentru sistemul de referinţă universal se foloseşte convenţia de sistem de coordonate drept. În grafica tridimensională se mai folosesc şi alte sisteme de referinţă, care permit descrierea operaţiilor de transformări geometrice şi care vor fi precizate pe parcursul lucrării.

Un punct P în spaţiul tridimensional se reprezintă în sistemul de referinţă Cartezian printr-un tripet de valori scalare, x, y, z, care reprezintă componentele vectorului de poziţie **OP** pe cele trei axe de coordonate. Dacă se noteză cu **i**, **j**, **k** versorii (vectorii unitate) ai celor trei axe de coordonate x, y, z, atunci vectorul de poziţie al punctului P este **OP** = x**i** + y**j** + z**k**. În notaţia matriceală, un punct în spaţiul tridimensional se poate reprezenta printr-o matrice linie sau printr-o matrice coloană:

$$\mathbf{P} = \begin{bmatrix} x & y & z \end{bmatrix} \text{ sau } \mathbf{P} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Ambele convenţii sunt folosite în egală măsură în sistemele grafice, ceea ce, din nou, poate provoca diferite confuzii, dacă nu se precizează convenţia folosită. Convenţia de reprezentare sub formă de matrice linie a unui punct, folosită în unele lucrări [Watt95], [Mold96], are avantajul că exprimă operaţiile de concatenare a matricelor într-un mod natural, de la stânga la dreapta. Convenţia de reprezentare matematică, standardul grafic PHIGS, biblioteca grafică OpenGL, ca şi unele din lucrările de referinţă în domeniu [Foley90] folosesc notaţia de matrice coloană pentru un punct în spaţiul tridimensional, care este adoptată şi în lucrarea prezentă.

2

Modelarea obiectelor

Modul cel mai convenabil de modelare a scenelor virtuale este acela în care fiecare obiect este modelat într-un sistem de coordonate propriu, numit sistem de referinţă model (sau sistem de referinţă local), în care punctele (vârfurile) obiectului sunt precizate relativ la un anumit punct de referinţă local. De fapt, în modelarea ierarhică, un obiect complex poate avea un număr oarecare de sisteme de referinţă locale, câte unul pentru fiecare parte componentă a sa. Instanţierea unui obiect în scena virtuală înseamnă amplasarea acestuia în sistemul de referinţă universal printr-o succesiune de scalări, rotaţii şi translaţii, care transformă obiectul din sistemul de referinţă local în sistemul de referinţă universal. Această succesiune de transformări este cunoscută sub numele de transformare de modelare.

Proprietăţile obiectelor tridimensionale care se modelează în aplicaţiile grafice se pot împărţi în două categorii: *forma* şi *atribute de aspect*. Informaţia de formă a unui obiect este diferită de celelalte atribute ale obiectului, deoarece forma este aceea care determină modul în care obiectul apare în redarea grafică şi toate celelalte atribute se corelează cu forma obiectului (de exemplu, culoarea se specifică pentru fiecare element de suprafaţă a obiectului).

Din punct de vedere al formei, obiectele tridimensionale reprezentate în grafica pe calculator pot fi obiecte *solide* sau obiecte *deformabile*. Un solid este un obiect tridimensional a cărui formă şi dimensiuni nu se modifică în funcţie de timp sau de poziţia în scenă (proprietatea de formă volumetrică invariantă). Majoritatea aplicaţiilor de realitate virtuală se bazează pe scene compuse din solide, dar există şi aplicaţii în care obiectele reprezentate îşi modifică forma şi dimensiunile într-un mod predefinit sau ca urmare a unor acţiuni interactive (de exemplu, în simulări ale

intervenţiilor chirurgicale). Chiar şi reprezentarea unor astfel de obiecte (obiecte deformabile) se bazează pe un model al unui solid, care se modifică în cursul experimentului de realitate virtuală. În lucrarea de faţă se vor prezenta modele ale solidelor, care stau la baza prelucrărilor din grafică şi realitate virtuală.

Modelarea solidelor este o tehnică de proiectare, vizualizare şi analiză a modului în care obiectele reale se reprezintă în calculator. În ordinea importanţei şi a frecvenţei de utilizare, metodele de modelare şi reprezentare a obiectelor sunt următoarele:

1. *Modelarea poligonală*. În acestă formă de reprezentare, obiectele sunt aproximate printr-o reţea de feţe care sunt poligoane planare.
2. *Modelarea prin petice parametrice bicubice* (*bicubic parametric patches*). Obiectele sunt aproximate prin reţele se elemente spaţiale numite petice (patches). Acestea sunt reprezentate prin polinoame cu două variabile parametrice, în mod obişnuit cubice.
3. *Modelarea prin combinarea obiectelor* (*Constructive Solide Geometry - CSG*). Obiectele sunt reprezentate prin colecţii de obiecte elementare, cum sunt cilindri, sfere, poliedre.
4. *Modelarea prin tehnica divizării spaţiale*. Obiectele sunt încorporate în spaţiu, prin atribuirea fiecărui element spaţial a unei etichete în funcţie de obiectul care ocupă elementul respectiv.

Aceste metode de modelare şi reprezentare a solidelor se pot grupa în reprezentări prin suprafaţa de frontieră (primele două metode) şi reprezentări prin volum (ultimele două metode).

2.1 Modelarea poligonală a obiectelor

Modelarea poligonală, în care un obiect constă dintr-o reţea de poligoane planare care aproximează suprafaţa de frontieră (*boundary representation – B-rep*), este forma "clasică" folosită în grafica pe calculator. Motivele utilizării extinse a acestei forme de reprezentare sunt uşurinţa de modelare şi posibilitatea de redare rapidă a imaginii obiectelor. Pentru obiectele reprezentate poligonal s-au dezvoltat algoritmi de redare eficienţi, care asigură calculul umbririi, eliminarea suprafeţelor ascunse, texturare, antialiasing, frecvent implementaţi hardware în sistemele grafice.

În reprezentarea poligonală, un obiect tridimensional este compus dintr-o colecţie de feţe, fiecare faţă fiind o suprafaţă plană reprezentată printr-un poligon.

2.1.1 Reprezentarea poligoanelor

Un poligon este o regiune din plan mărginită de o colecţie finită de segmente de dreaptă care formează un circuit închis simplu.

Fie n puncte în plan, notate v_0, v_1, …,v_{n-1} şi n segmente de dreaptă $e_0 = v_0v_1$, $e_1 = v_1v_2$, …. $e_{n-1} = v_{n-1}v_0$, care conectează perechi de puncte succesive în ordine ciclică, deci inclusiv conexiunea între ultimul punct şi primul punct din listă. Aceste segmente mărginesc un poligon dacă şi numai dacă:

(a) Intersecţia fiecărei perechi de segmente adiacente în ordinea ciclică este un singur punct, conţinut de ambele segemente: $e_i \cap e_{i+1} = v_{i+1}$, pentru oricare $i = 0,\ldots,n-1$.

(b) Segmente neadiacente nu se intersectează: $e_i \cap e_j = \emptyset$, pentru orice $j \neq i+1$.

Segmentele care mărginesc un poligon (linia poligonală) formează un circuit închis (ciclu) deoarece segmentele sunt conectate capăt la capăt şi ultimul segment conectează ultimul punct cu primul punct; ciclul este simplu deoarece segmentele neadiacente nu se intersectează.

Punctele v_i se numesc vârfurile poligonului (*vertices*); segmentele e_i se numesc muchii (sau laturi) ale poligonului. De remarcat că un poligon conţine n vârfuri şi n muchii şi că muchiile sunt orientate, astfel încât formează un ciclu (circuit închis). O astfel de orientare a segmentelor se numeşte orientare consistentă. În general, se foloseşte ordinea de parcurgere în sensul invers acelor de ceasornic: dacă se parcurg muchiile în sensul lor de definiţie, interiorul poligonului este văzut întotdeauna în partea stângă (Fig. 2.1).

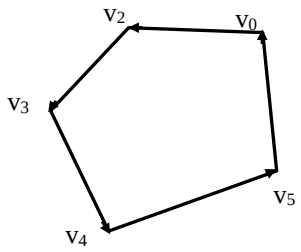


Fig. 2.1 Segmentele liniei poligonale sunt orientate şi nu se autointerectează.

O teoremă importantă în prelucrarea poligoanelor este teorema lui *Jordan*, care spune că orice curbă plană închisă simplă împarte planul în două părţi: o parte interioară curbei, care este o regiune limitată, şi o parte exterioară curbei, care este o regiune nelimitată. Deşi pare simplă din punct de vedere intuitiv, demonstraţia teoremei lui Jordan este destul de dificilă şi poate fi găsită în referinţele bibliografice [Rour93].

Această teoremă justifică definiţia care se mai foloseşte pentru poligoane, şi anume: poligonul este o regiune limitată din plan, mărginită de o colecţie de segmente orientate consistent. Prin această definiţie se consideră poligonul ca o regiune închisă din plan. Uneori, poligonul este considerat ca fiind format numai din conturul său, deci numai de segmentele de dreaptă care mărginesc regiunea, şi nu de regiunea însăşi. În continuare, se foloseşte notaţia ∂P pentru a desemna conturul poligonului P (linia poligonală închisă care mărgineşte poligonul).

Triangularizarea poligoanelor. O altă proprietate importantă a poligoanelor folosită în grafică este proprietatea de triangularizare. Se demonstrează că orice poligon poate fi împărţit în triunghiuri prin adăugarea a zero sau mai multe diagonale. Proprietatea de triangularizare se bazează pe noţiunile de vizibilitate şi diagonală în poligoane.

Un punct x din interiorul unui poligon poate vedea un alt punct y, dacă şi numai dacă segmentul xy nu este în nici un punct al său exterior poligonului, adică $xy \subseteq P$. Acest lucru însemnă că linia care uneşte două puncte x şi y, vizibile unul altuia, poate atinge un vârf al poligonului. Vizibilitatea între două puncte x şi y este completă (*clearly visible*), dacă linia care uneşte cele două puncte nu atinge frontiera poligonală (Fig. 2.2).

O diagonală a unui poligon este un segment de dreaptă între două vârfuri a şi b, complet vizibile unul altuia. Acest lucru înseamnă că intersecţia dintre segmentul închis ab şi ∂P este mulţimea {a,b}, adică segmentul ab nu atinge linia poligonală ∂P în alte puncte decât vârfurile a şi b, de început şi de sfârşit ale segmentului. Condiţiile ca segmentul ab care uneşte vârfurile a şi b ale unui poligon să fie o diagonală în acel poligon sunt deci: $ab \subseteq P$ şi $ab \cap \partial P = \{a, b\}$. Orice diagonală împarte un poligon în două poligoane mai mici (Fig. 2.2).

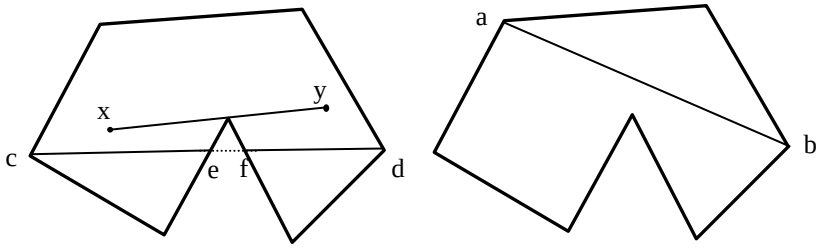


Fig. 2.2 Vizibilitate şi diagonale în poligoane.
Punctele x şi y sunt vizibile unul altuia.
Punctele c şi d sunt invizibile unul altuia deoarece $ef \not\subset P$.
Punctele a şi b sunt complet vizibile unul altuia, deci ab este diagonală în P.

Două diagonale ale unui poligon sunt neîncrucişate (*noncrossing*) dacă intersecţia lor este o submulţime a capetelor lor (punctele de început şi de sfârşit ale segmentelor).

Dacă se adaugă atâtea diagonale neîncrucişate câte sunt posibile într-un poligon, atunci poligonul este împărţit în triunghiuri. O astfel de partiţionare a unui poligon în triunghiuri se numeşte *triangularizarea* poligonului. Diagonalele se pot adăuga în orice ordine, atâta timp cât sunt neîncrucişate. Demonstraţia teoremei conform căreia orice poligon admite o triangularizare, se bazează pe teorema lui Meister, demonstrată în [Rour93], care stabileşte că orice poligon cu $n \geq 4$ vârfuri admite cel puţin o diagonală.

Teorema triangularizării se bazează şi pe lema numărului de diagonale: Orice triangularizare a unui poligon P cu n vârfuri utilizează $n - 3$ diagonale şi constă din $n - 2$ triunghiuri. Aceste teoreme se demonstrează prin inducţie. În Fig. 2.3 este prezentată triangularizarea unui poligon convex cu opt laturi; se inserează 8 – 3 = 5 diagonale neîncrucişate şi rezultă 8 – 2 = 6 triunghiuri.

Teorema triangularizării, care asigură că orice poligon poate fi divizat în triunghiuri, reprezintă suportul celei mai eficiente metode de generare a imaginii (redarea) obiectelor tridimensionale: obiectele se reprezintă prin feţe poligonale,

fiecare poligon se descompune în triunghiuri şi triunghiurile sunt generate prin algoritmi implementaţi hardware.

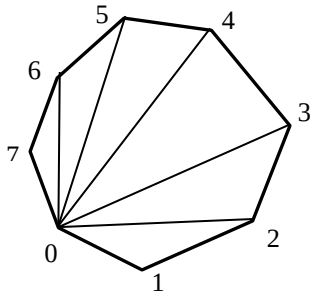


Fig. 2.3 Triangularizarea unui poligon convex.

Din punct de vedere al reprezentării în program a poligoanelor, cea mai compactă formă este reprezentarea printr-o listă liniară de vârfuri, fiecare vârf fiind specificat printr-o structură (sau clasă, în programarea orientată pe obiecte) care memorează (cel puţin) coordonatele vârfului. Alte date referitoare la vârfurile poligoanelor necesare în modelarea şi redarea obiectelor (normală, culoare, coordonate de texturare, etc.) vor fi descrise în secţiunile care urmează.

Este posibilă reprezentarea unui poligon şi prin lista segmentelor sale, dar acestă reprezentare necesită un volum mai mare de date şi este folosită în implementarea anumitor algoritmi de prelucrare a poligoanelor (reuniune, divizare, etc.) şi mai puţin în reprezentarea modelului unui obiect. Lista liniară de vârfuri poate fi implementată ca vector sau ca listă simplu sau dublu înlănţuită.

2.1.2 Reprezentarea poliedrelor

În modelarea şi reprezentarea prin suprafaţa de frontieră, obiectele sunt aproximate prin poliedre şi modelul lor este reprezentat prin suprafaţa poliedrului, compusă dintr-o colecţie de poligoane.

Un poliedru reprezintă generalizarea în spaţiul tridimensional a unui poligon din planul bidimensional: poliedrul este o regiune finită a spaţiului a cărui suprafaţă de frontieră este compusă dintr-un număr finit de feţe poligonale plane. Suprafaţa de frontieră a unui poliedru conţine trei tipuri de elemente geometrice: vârfurile (punctele) care sunt zero-dimensionale, muchiile (segmentele), care sunt unidimensionale şi feţele (poligoane) care sunt bidimensionale (Fig. 2.4).

Suprafaţa de frontieră a unui poliedru este o colecţie finită de feţe poligonale care se intersectează corect. Intersecţia corectă a feţelor înseamnă că pentru fiecare pereche de feţe ale obiectului, feţele sunt disjuncte, au în comun un singur vârf, sau au în comun două vârfuri şi muchia care le uneşte.

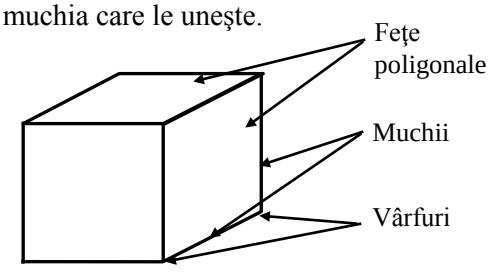


Fig. 2.4 Reprezentarea prin suprafaţa de frontieră a unui poliedru.

Din punct de vedere matematic, nu este imediat evident că un solid poate fi reprezentat univoc prin suprafaţa care îl mărgineşte. De aceea, este necesar să fie stabilite condiţiile în care această reprezentare este permisă. Aceste condiţii, numite condiţii de *construcţie corectă*, se definesc pentru suprafeţe de frontieră triangularizate. Triangularizarea unei suprafeţe poliedrale se obţine prin triangularizarea fiecărei feţe poligonale, astfel încât suprafaţa rezultată constă din vârfuri care sunt înconjurate de triunghiuri, fiecare pereche de triunghiuri fiind adiacente de-a lungul unei muchii. Laturile triunghiurilor adiacente unui vârf formează un circuit de segmente, numit link-ul vârfului (Fig. 2.5).

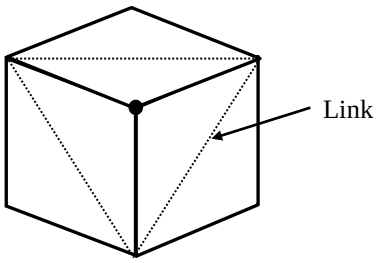


Fig. 2.5 Triangularizarea suprafeţei de frontieră.
Link-ul unui vârf al suprafeţei.

O suprafaţă de frontieră construită corect îndeplineşte următoarele condiţii:

- Linkul fiecărui vârf al suprafeţei triangularizate este complet, adică formează un circuit închis, nu neapărat planar.
- Triunghiurile suprafeţei triangularizate sunt orientate consistent.

Închiderea link-ului fiecărui vârf asigură proprietăţile suprafaţei de a fi *închisă* şi *conectată*. Proprietate de închidere înseamnă că suprafaţa nu are un sfârşit. Proprietatea de conectivitate înseamnă că există cel puţin o cale de la un triunghi la altul aparţinând aceleiaşi suprafeţe de frontieră, care poate fi parcursă prin traversarea muchiilor adiacente. Dacă o suprafaţă nu este închisă sau nu este conectată, prin triangularizarea suprafeţei nu se obţin link-uri închise.

Proprietate de construcţie corectă a suprafeţei de frontieră a unui solid se poate verifica folosind condiţia de *orientare consistentă* a suprafeţei, formulată de legea lui Moebius: O suprafaţă închisă este orientată consistent dacă la traversarea triunghiurilor sale (rezultate dintr-o triangularizare arbitrară), într-o direcţie unică (de exemplu, în direcţia inversă acelor de ceasornic), fiecare muchie este traversată de două ori, în direcţii opuse (Fig. 2.6).

Generalizarea în spaţiul tridimensional a teoremei lui Jordan spune că orice suprafaţă de frontieră închisă, consistent orientată, împarte spaţiul în două părţi: o

parte interioară suprafeţei, care este o regiune limitată, şi o parte exterioară suprafeţei, care este o regiune nelimitată.

Din punct de vedere geometric, orientarea consistentă se verifică prin direcţia normalelor la feţele obiectului: dacă normalele feţelor sunt îndreptate către aceeaşi regiune a spaţiului (fie toate îndreptate spre interior, fie toate îndreptate spre exterior), atunci suprafaţa are o orientare consistentă. Acest mod de verificare se referă la obiectele tridimensionale fără cavităţi, dar se poate extinde cu uşurinţă şi la obiecte care prezintă cavităţi.

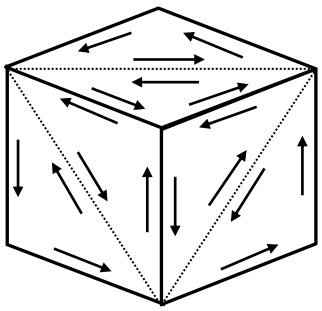


Fig. 2.6 Legea lui Mobius verifică orientarea consistentă a unei suprafeţe triangularizate.

Teoretic, orientarea consistentă se verifică pentru feţele triangularizate ale suprafeţei de frontieră, dar, prin extindere, se poate verifica orientarea consistentă folosind normalele la feţele poligonale, deoarece toate triunghiurile obţinute prin triangularizarea unui poligon care reprezintă o faţă a unui obiect au aceeaşi orientare (Fig. 2.7).

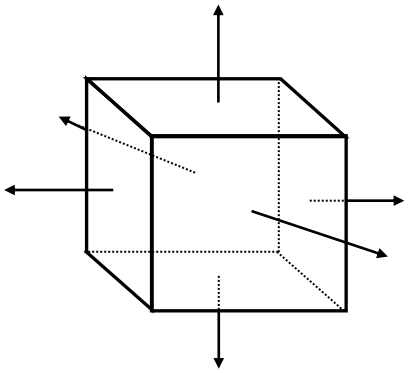


Fig. 2.7 Orientarea consistentă a feţelor obiectului.

Orientarea consistentă a feţelor poligonale ale obiectelor este o condiţie de verificare a construcţiei corecte a suprafeţei de frontieră şi, în acelaţi timp, este folosită în operaţiile de eliminare a suprafeţelor ascunse în cursul redării obiectelor tridimensionale. De aceea, normalele la feţele poligonale se memorează în modelul obiectelor, împreună cu coordonatele vârfurilor.

2.1.3 Implementarea modelului poligonal

Reprezentarea prin reţea de poligoane a obiectelor se implementează printr-o listă de poligoane, care poate fi un vector sau o listă înlănţuită. Astfel, întreaga informaţie referitoare la forma unui obiect este compusă din liste de coordonate ale vârfurilor, la care se mai adaugă şi alte informaţii geometrice necesare în redare (de exemplu, normalele la suprafeţe).

Fără să se insiste acum asupra eficienţei sau eleganţei implementării (se va reveni ulterior), se poate considera că forma unui obiect modelat prin reţea de poligoane poate fi reprezentată ca un vector de feţe, fiecare faţă fiind un vector de vârfuri, fiecare vârf fiind un vector de trei coordonate în virgulă flotantă. De exemplu, un cub poate fi reprezentat astfel:

```
double CubeFaces[6][4][3] = {
        {{-1,-1,-1},                    // fata 0
         { 1,-1,-1},
         { 1,-1, 1},
         {-1,-1, 1}},
        {{-1, 1, 1},          // fata 1
         { 1, 1, 1},
         { 1, 1,-1},
         {-1, 1,-1}},
        {{-1,-1, 1},          // fata 2
         { 1,-1, 1},
         { 1, 1, 1},
         {-1, 1, 1}},
        {{ 1,-1,-1},                    // fata 3
         {-1,-1,-1},
         {-1, 1,-1},
         { 1, 1,-1}},
  {{ 1,-1, 1},                  // fata 4
         { 1,-1,-1},
         { 1, 1,-1},
         { 1, 1, 1}},
  {{-1,-1, 1},                  // fata 5
         {-1, 1, 1},
         {-1, 1,-1},
         {-1,-1,-1}}
};
```

O astfel de implementare este însă ineficientă deoarece fiecare vârf este prelucrat de trei ori, pentru fiecare faţă adiacentă acestuia. O implementare mult mai eficientă a modelului poligonal defineşte un vector cu toate vârfurile unui obiect, iar fiecare faţă se defineşte printr-un vector de indecşi în vectorul de vârfuri. De exemplu, modelul unui cub folosind indecşi pentru definirea feţelor arată astfel:

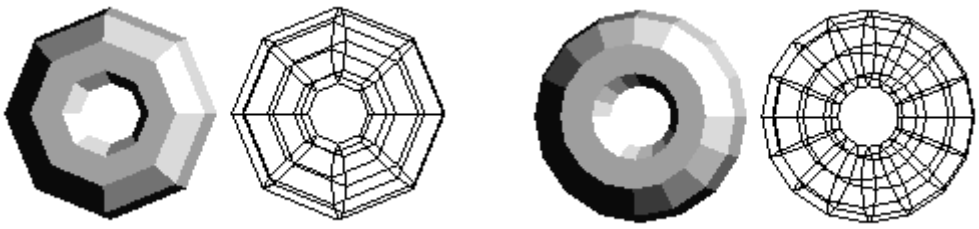
```
double CubeCoords[8][3]={
        {-1,-1, 1},
        { 1,-1, 1},
        { 1,-1,-1},
        {-1,-1,-1},
        {-1, 1, 1},
        { 1, 1, 1},
        { 1, 1,-1},
        {-1, 1,-1}
};
```

```
int CubeIndexFace[6][4]={
        {3, 2, 1, 0},
        {4, 5, 6, 7},
        {0, 1, 5, 4},
        {2, 3, 7, 6},
        {1, 2, 6, 5},
        {0, 4, 7, 3}
};
```

Atât în cursul modelării, cât şi în cursul redării imaginii, modelul obiectelor se reprezintă mai complex, conţinând şi alte informaţii geometrice şi atribute de aspect. Astfel, culoarea (mai general spus, materialul) este o informaţie care se asignează fiecărei feţe sau fiecărui vârf al obiectului. În general, modelul poligonal cu reprezentarea prin indecşi a feţelor poligonale este implementat orientat pe obiecte, folosind mai multe clase (de bază şi derivate) care permit încapsularea tuturor informaţiilor necesare pentru redarea obiectelor.

O reţea de poligoane poate să reprezinte fie suprafaţa de frontieră a unui solid, fie o suprafaţă deschisă, necesară în anumite situaţii de modelare, cum este suprafaţa terenului simulat într-o scenă virtuală. Deoarece se modelează o zonă geografică limitată şi nu se explorează scena astfel ca să fie văzută marginea "pământului", se poate folosi o suprafaţă poligonală deschisă pentru reprezentarea terenului.

Reprezentarea obişnuită în grafica din scenele virtuale este aceea în care fiecare poligon este reprezentat ca o suprafaţă (poligoane "pline"), dar în proiectările grafice (în mecanică, construcţii, etc) se foloseşte şi reprezentarea prin contur a poligoanelor, numită reprezentare "cadru de sârmă" (*wireframe*) (Fig. 2.8).



(a) (b)

Fig. 2.8 Reprezentarea "plină" (*filled*) şi "cadru de sârmă" (*wireframe*) a obiectelor.

Precizia de reprezentare a modelului, adică diferenţa dintre suprafaţa obiectului şi feţele poligonale prin care este aproximat, este un parametru important de modelare. În general, cu cât numărul de poligoane prin care se aproximează suprafaţa obiectului este mai mare, cu atât precizia de reprezentare este mai bună. Se observă diferenţa dintre imaginea unui obiect (tor) reprezentat prin 64 de poligoane în Fig. 2.8(a) şi prin 128 de poligoane în Fig. 2.8 (b).

Folosirea unui număr mare de poligoane pentru reprezentarea obiectelor conduce la un volum imens de date în modelarea scenelor virtuale, care implică cerinţe corespunzătoare de memorare şi de capacitate de calcul. Oricât de mult ar creşte capacitatea de prelucrare în sistemele grafice, prin folosirea paralelismului şi

prin implementarea hardware a algoritmilor de prelucrare, performanţe de generare interactivă în timp real a imaginilor se pot obţine numai dacă obiectele scenei se modelează în mod corespunzător. Metodele de aproximare eficientă a modelelor sunt cunoscute sub numele de *simplificarea* (sau *rafinarea*) *datelor*.

Tehnica de simplificare directă, prin reducerea uniformă a numărului de poligoane de reprezentare a obiectului este inacceptbilă, deoarece nu se poate controla precizia de reprezentare. Pentru păstrarea unei precizii de reprezentare constantă, se foloseşte modelarea adaptivă, în care dimensiunea feţelor poligonale variază în funcţie de curbura suprafeţei; în părţile cu curbură pronunţată sunt introduse mai multe poligoane pe unitatea de suprafaţă.

O altă metodă se simplificare a datelor este modelarea cu nivele de detaliu multiple a obiectelor (*levels of detail*- LOD). Fiecare model al unui obiect este compus dintr-o succesiune de reprezentări, fiecare cu o precizie diferită. În cursul generării imginilor, se selectează nivelul de detaliu corespunzător, în funcţie de poziţia obiectului faţă de punctul de observare. Cu cât obiectul este mai depărtat, se selectează o reprezentare cu o precizie mai scăzută a obiectului. Acestă metodă este posibilă datorită folosirii proiecţiei perspectivă în generarea imaginilor şi va fi înţeleasă mai uşor după parcurgerea secţiunii următoare privind transformărilor geometrice.

Modelul poligonal al unui obiect se poate genera prin mai multe metode, în funcţie de tipul obiectului şi de aplicaţia grafică în care este folosit modelul respectiv. Se poate folosi una din următoarele metode de modelare poligonală:

- Generarea modelului din descrierea matematică a obiectului.
- Generarea modelului obiectului prin baleiere spaţială.
- Generarea modelului pornind de la o mulţime de puncte care aparţin suprafeţei de frontieră a obiectului.

2.1.4 Generarea modelului din descrierea matematică

Se poate genera reţeaua de poligoane de aproximare a obiectelor care au o descriere matematică cunoscută. De exemplu, ecuaţiile de definiţie a unor suprafeţe quadrice:

- Elipsoid:

$$x^2/a^2 + y^2/b^2 + z^2/c^2 - 1 = 0 \qquad (2.1)$$

unde a, b c sunt semiaxele elipselor.

- Hiperboloid:

$$x^2/a^2 + y^2/b^2 - z^2/c^2 - 1 = 0 \quad \text{si} \quad x^2/a^2 + y^2/b^2 - z^2/c^2 + 1 = 0 \qquad (2.2)$$

- Paraboloid eliptic:

$$x^2/a^2 + y^2/b^2 = z \qquad (2.3)$$

Surprafaţa se intersectează mai întâi cu un număr n de plane perpendiculare pe axa Oz, de ecuaţii $z = n\ \Delta z$, pentru n = − k, −k+1, −1, 0, 1 , 2 ,...k−1, k. Se obţin n elipse (paralele) şi pe fiecare elipsă se eşantionează m puncte echidistante (pe meridiane), obţinându-se (n −1) x m poligoane care aproximează suprafaţa de frontieră a obiectului respectiv.

Aceste suprafeţe se pot obţine şi prin rotaţia unei curbe în jurul unei axe de coordonate. De exemplu, suprafaţa elipsoidului se obţine prin rotaţia în jurul axei z a elipsei:

$$x^2/a^2 + y^2/b^2 + z^2/c^2 - 1 = 0 \qquad (2.4)$$
$$y = 0.$$

Prin rotaţia unei curbe în jurul unei axe se pot obţine obiecte tridimensionale mai variate, în funcţie de forma curbei care se roteşte. De exemplu, un tor se obţine prin rotaţia unui cerc în jurul unei axe paralele cu planul cercului. Suprafeţele astfel obţinute se numesc suprafeţe de rotaţie.

2.1.5 Generarea modelului prin baleiere spațială

Se pot genera obiecte tridimensionale prin deplasarea (*sweeping*) unei suprafeţe generatoare de-a lungul unei curbe oarecare. Dacă se variază forma şi orientarea suprafeţei generatoare în cursul deplasării, se pot obţine obiecte variate, în funcţie de forma curbei şi de orientarea, forma şi variaţia formei suprafeţei generatoare. Prin această metodă se pot obţine atât formele regulate descrise mai sus (elipsoid, hiperboloid, paraboloid eliptic, tor) cât şi alte obiecte numite solide ductibile sau extrudate (*ducted solids*) sau cilindri generalizaţi (*generalized cylinders*) (Fig. 2.9).

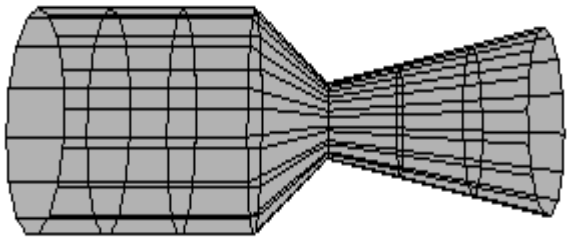


Fig. 2.9 Obiect poligonal modelat prin deplasarea unui cerc de diametru variabil de-a lungul unei linii drepte.

Pentru definirea deplasării unei suprafeţe de-a lungul unei curbe, este necesar să se definească intervalul u al curbei pe care are loc deplasarea şi modul în care se divide intervalul parcurs. Împărţirea intervalului în distanţe egale nu dă rezultate bune, deoarece punctele obţinute nu vor fi egal distribuite pe suprafaţa obiectului. De aceea este necesară divizarea intervalului în funcţie de curbura curbei. Dacă curbura este

pronunţată, se aleg subdiviziuni mai mici ale intervalului, iar pentru curburi mai reduse se aleg subdiviziuni mai mari ale intervalului.

2.1.6 Generarea modelului pornind de la o mulțime de puncte care aparțin suprafeței de frontieră a obiectului

O metodă mai generală de modelare a obiectelor tridimensionale se bazează pe cunoaşterea unei mulţimi de puncte distribuite uniform sau neuniform pe suprafaţa obiectului. Această metodă implică mai întâi obţinerea colecţiei de puncte, şi apoi construirea reţelei de poligoane care să aproximeze obiectul cu precizia dorită. Dacă metodele prezentate mai înainte pot fi utilizate pentru o categorie restrânsă de obiecte, în schimb, metoda generării modelului pornind de la o mulţime de puncte ale suprafeţei acestuia poate fi aplicată pentru cele mai variate obiecte: clădiri, vehicule, plante, animale, elemente anatomice, teren, etc.

Mulţimea de puncte de pe suprafaţa obiectului se determină diferit, în funcţie de modul în care este cunoscut sau reprezentat obiectul real:

- Pentru obiectele care se proiectează într-un sistem de proiectare bazat pe calculator – CAD (*Computer Aided Design*), informaţiile de formă necesare în generearea modelului se pot obţine din reprezentarea proiectului. De exemplu, clădiri, obiecte mecanice, vehicule, pot fi proiectate în AutoCad şi coordonatele vârfurilor suprafeţelor sunt folosite pentru reprezentarea prin reţea poligonală a obiectului.
- Pentru obiectele real existente, sau machete ale acestora, se poate folosi un scanner 3D cu laser. Obiectul real (sau macheta acestuia) este plasat pe o masă rotativă în direcţia de emisie a unei raze laser. La fiecare rotaţie completă a mesei se obţine o colecţie de puncte pe un contur al obiectului, prin măsurarea distanţei la suprafaţa obiectului. După fiecare rotaţie, se deplasează masa în sus sau în jos, astfel că se obţine o colecţie de contururi ale obiectului. Toate aceste puncte de pe suprafaţa obiectului sunt apoi folosite pentru crearea modelului poligonal.
- Pentru modelarea terenului se folosesc hărţi digitale care furnizează altitudinile terenului într-o grilă uniformă de puncte, sau contururi de nivel, care unesc puncte cu altitudine constantă.

Fiind cunoscută o mulţime de puncte V care caracterizează un obiect tridimensional, volumul $\Omega \subset R^3$ acoperit de aceste puncte se numeşte domeniul mulţimii V. Ω este un poliedru, convex sau neconvex, iar punctele mulţimii V pot fi dispuse regulat sau neregulat în domeniul Ω. Cazul cel mai frecvent întâlnit în modelare este acela în care punctele mulţimii V sunt distribuite neregulat în spaţiu şi aproximarea obiectului se realizează printr-o reţea tridimensională de poligoane Γ, care aproximează linear pe porţiuni obiectul dat. Reţeaua Γ trebuie să fie construită pornind de la mulţimea V de puncte în spaţiu şi de la cerinţele de precizie de aproximare a obiectului.

Construirea modelului poligonal al unui obiect cunoscut printr-o mulţime de puncte de pe suprafaţa sa se poate realiza prin mai multe metode, dintre care cele mai folosite sunt triangularizarea (*triangulation*) şi transformata Wavelet.

2.1.7 Redarea imaginii obiectelor poligonale

Aşa cum s-a mai amintit, modelarea poligonală este cea mai folosită formă de modelare a obiectelor în grafica pe calculator, datorită simplităţii reprezentării modelului poligonal şi a faptului că, în momentul de faţă, majoritatea acceleratoarelor din sistemele grafice conţin programe implementate hardware de redare eficientă a poligoanelor. Algoritmii de redare implementaţi în acceleratoarele grafice tratează separat fiecare poligon (primitivă geometrică) reprezentat prin coordonatele vârfurilor sale, ceea ce face ca procesul de redare a imaginii să fie foarte simplu. Redarea imaginilor scenelor virtuale compuse din obiecte a căror reprezentare însumează mii de poligoane nu este ceva neobişnuit.

Redarea obiectelor poligonale este implementată printr-o succesiune de operaţii de transformări grafice asupra obiectelor, numită *pipeline grafic*. Această succesiune, care va fi descrisă pe larg în secţiunea următoare, constă din transformări geometrice aplicate vârfurilor obiectelor, prin care se transformă fiecare faţă a obiectului din sistemul de referinţă de modelare într-un sistem de referinţă de afişare, urmată de transformarea de redare, prin care se obţine culoare pixelilor care se afişează pe display. În Fig. 2.10 este redată imaginea avionului F-16 modelat prin 2366 feţe poligonale şi redat prin prelucrarea fiecărei feţe, în modul cu feţe "pline" şi umbrire şi în modul wireframe.

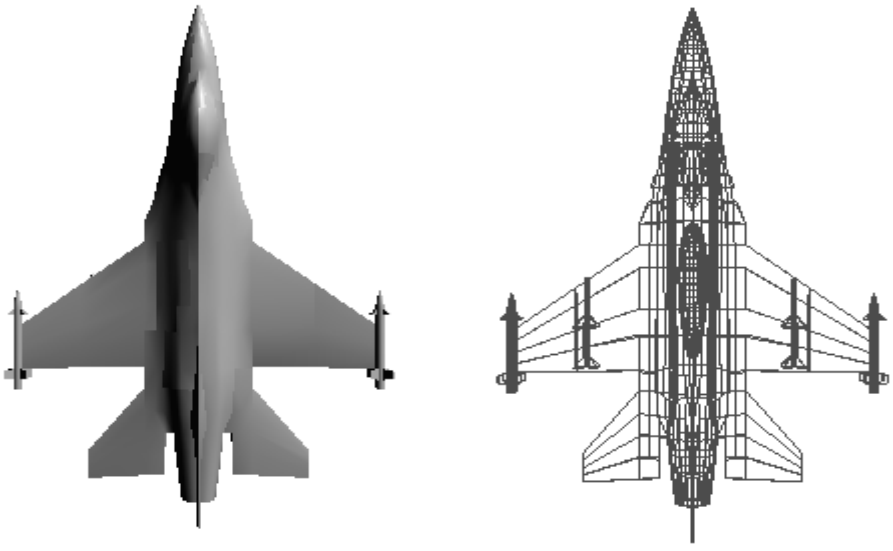


Fig. 2.10 Redarea obiectelor modelate prin reţea de feţe poligonale.

Pentru toate celelalte modele de reprezentare a obiectelor (reprezentarea prin reţele de petice, prin compunerea obiectelor sau prin subdivizarea spaţială), pe lângă redarea directă a modelului respectiv, există şi posibilitatea de redare prin deducerea

mai întâi a reprezentării poligonale corespunzătoare, urmată de folosirea algoritmilor de redare poligonală.

2.2 Modelarea obiectelor prin reţele de petice

Un petic (*patch*) este o suprafaţă curbă definită parametric în spaţiul tridimensional. Prin reprezentarea parametrică, punctele de pe suprafaţă se pot calcula secvenţial, pentru diferite valori ale parametrilor, mult mai simplu decât prin rezolvarea sistemului de ecuaţii care descriu implicit suprafaţa. Reprezentarea parametrică a curbelor şi a suprafeţelor este folosită în proiectarea şi în modelarea obiectelor pentru obţinerea unei precizii mai ridicate de aproximare.

Un segment de curbă în spaţiul tridimensional poate fi definit printr-un sistem de ecuaţii în funcţie de un parametru:

$$\begin{aligned} x(u) &= a_x u^3 + b_x u^2 + c_x u + d_x \\ y(u) &= a_y u^3 + b_y u^2 + c_y u + d_y \\ z(u) &= a_z u^3 + b_z u^2 + c_z u + d_z \end{aligned} \tag{2.5}$$

Aceasta este o curbă cubică, definită ca formă şi mărime de cei 12 coeficienţi, numiţi coeficienţi algebrici ai curbei. În notaţie vectorială se poate scrie forma parametrică a unei curbe cubice:

$$\mathbf{Q}(u) = \mathbf{A}u^3 + \mathbf{B}u^2 + \mathbf{C}u + \mathbf{D} \tag{2.6}$$

unde vectorul **Q** are componentele (x, y, z), iar vectorii **A**, **B**, **C**, **D** au componentele (a_x, a_y, a_z), (b_x, b_y, b_z), (c_x, c_y, c_z), (d_x, d_y, d_z), respectiv, iar u este cuprins în intervalul închis [0,1].

În aplicaţiile grafice se utilizează în mod frecvent curbele cubice, deoarece ele asigură suficientă flexibilitate în definirea curbelor şi pot fi prelucrate eficient. Polinoame de grad mai mare pot descrie curbe mai complexe, dar necesită un număr mai mare de coeficienţi şi sunt mai dificil de prelucrat eficient în reprezentările grafice. La modul general, o curbă parametrică cubică se poate defini prin ecuaţiile:

$$\mathbf{Q}(u) = \sum_{i=0}^{3} \mathbf{P}_i \mathbf{B}_i(u) \tag{2.7}$$

unde **Q** este vectorul de componente (x, y, z), $\mathbf{P}_i$ sunt punctele de control ale curbei, fiecare vector $\mathbf{P}_i$ având componentele (x_i, y_i, z_i), iar $\mathbf{B}_i$ sunt funcţiile de bază (sau de amestec – *blending functions*) ale reprezentării parametrice.

Diferite funcţii de bază au proprietăţi diferite în determinarea formei curbei parametrice: ele pot interpola sau aproxima o curbă dată, pot asigura anumite condiţii de continuitate a mai multor segmente de curbă.

Specificarea unui segment de curbă (sau a unui petic de suprafaţă curbă) printr-un set de puncte de control este o metodă de bază în proiectarea grafică interactivă: proiectantul defineşte punctele de control; curba este generată şi vizualizată interactiv; dacă forma curbei nu este mulţumitoare, proiectantul modifică unul sau mai multe puncte de control, până obţine rezultatul dorit.

Cele mai utilizate tipuri de curbe şi suprafeţe parametrice în proiectarea grafică sunt curbele (şi suprafeţele) Bézier şi B-spline. În bibliotecile grafice există funcţii de generare a curbelor şi suprafeţelor parametrice.

2.3 Modelarea prin compunerea obiectelor

Modelarea prin compunerea obiectelor (*Constructive Solid Geometry* – CSG) se foloseşte atunci când un obiect poate fi obţinut prin combinarea mai multor obiecte elementare, numite primitive geometrice.

Primitivele geometrice utilizate sunt sfere, conuri, cilindri sau paralelipipede dreptunghice şi sunt combinate folosind operatori booleni şi transformări liniare. Un obiect complex este reprezentat printr-un arbore, ale cărui frunze sunt primitivele geometrice iar nodurile memorează operatori booleeni sau transformări liniare.

În Fig. 2.11 este prezentată operaţia de reuniune a două obiecte elementare. Alte operaţiile posibile care se pot efectua asupra primitivelor geometrice sunt scăderea şi intersecţia.

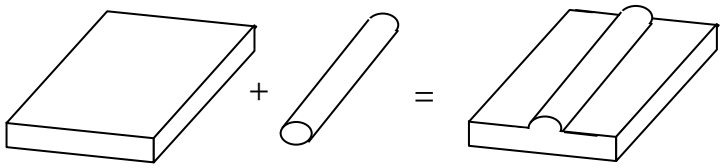


Fig. 2.11 Operaţia de reuniune a două primitive geometrice.

Fig. 2.12 arată reprezentarea CSG a unui obiect prin combinarea a trei obiecte elementare: două paralelilpipede dreptunghice şi un cilindru. Arborele de reprezentare are ca frunze primitivele geometrice, iar celelate noduri conţin operatorii booleni. Paralelipipedele sunt combinate folosind operaţia de reunuine, iar o gaură este practicată într-unul din paralelipipede prin scăderea unui cilindru din ansamblul celor două paralelipipede.

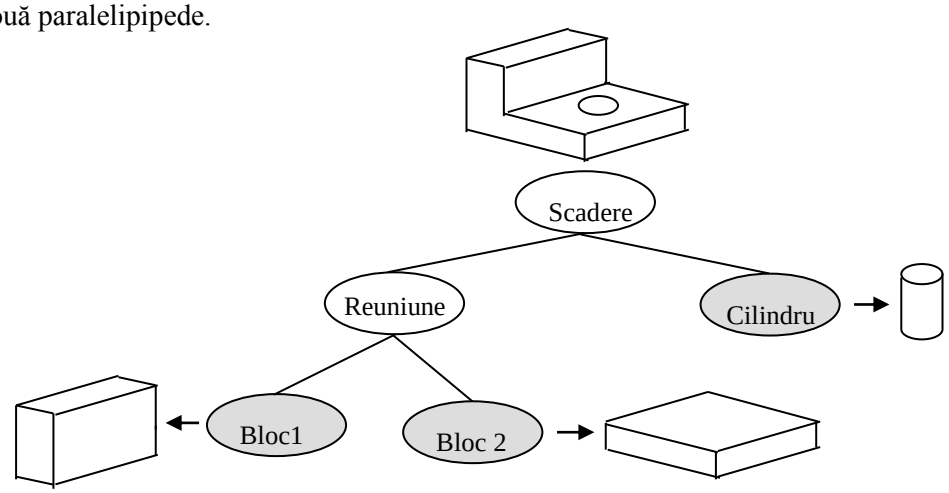


Fig. 2.12 Arborele de construire a unui obiect din trei primitive geometrice.

Deoarece arborele de reprezentare memorează atât operaţiile boolene cât şi forma primitivelor geometrice, operaţiile de modificare ale obiectului compus sunt relativ simple. De exemplu, o gaură într-un obiect se modifică prin modificarea poziţiei sau a dimensiunii primitivei geometrice folosite în operaţia de scădere. O astfel de modificare este mult mai dificilă în reprezentarea prin suprafaţa de frontieră a modelelor.

Redarea imaginii obiectelor CSG se poate face prin mai multe metode:

- Redarea directă a obiectului în aplicaţiile de *ray-tracing*.
- Conversia la reprezentarea prin suprafaţa de frontieră (B-rep) pornind de la modelul CSG şi aplicarea procedurilor standard de redare a poligoanelor.
- Conversia la reprezentarea prin subdivizarea spaţiului (în elemente numite voxeli) şi redarea volumului corespunzător

Tehnica ray-tracing generează imaginea obiectelor prin intersecţia acestora cu raze de proiecţie şi permite obţinerea unor efecte deosebit de realiste în modelarea reflexiei, transparenţei şi a umbrelor.

Un dezavantaj important al modelării CSG este cela că nu orice fel de obiect poate fi obţinut prin combinarea unor primitive geometrice simple. De exemplu, modelul unei statui (folosit în crearea muzeelor virtuale) sau modelul unui organ anatomic (folosit în simulările de intervenţii chirurgicale) nu pot fi obţinute prin combinarea unor primitive geometrice simple.

2.4 Modelarea prin divizarea spaţială

În tehnica de divizare spaţială, se atribuie fiecărei subdiviziuni a spaţiului tridimensional câte o etichetă în funcţie de obiectul care ocupă acea subdiviziune.

În acestă tehnică se consideră spaţiul tridimensional compus dintr-un număr de m x n x k volume elementare (numite *voxeli*), şi pentru fiecare voxel se memorează eticheta (un număr de identificare) obiectului care ocupă acel element. Reprezentarea prin subdiviziunea spaţiului este utilă în diferite aplicaţii grafice, cum sunt imagistica medicală sau aplicaţiile de *ray tracing*.

Implementarea directă, prin atribuirea unei etichete fiecărui element de volum al spaţiului, implică o mare redundanţă a datelor: toţi voxelii interiori unui obiect conţin acceaşi informaţie, eticheta obiectului. Acest lucru conduce la o cantitatate de informaţie foarte mare necesară pentru reprezentarea unui spaţiu de obiecte şi este, în general, inacceptabilă.

Pentru reducerea redundanţei de reprezentare se foloseşte o tehnică de reprezentare a ocupării spaţiale printr-un arbore numit arbore octal (*octree*). Arborele octal este o structură ierarhică care specifică ocuparea unei regiuni cubice din spaţiul tridimensional. Ideea de reprezentare a ocupării spaţiale poate fi exemplificată mai uşor într-un spaţiu bidimensional prin intermediul unui arbore cuaternar (*quadtree*). În Fig. 2.13(a) este prezentată ocuparea unei regiuni bidimensionale de trei obiecte, iar în Fig. 2.14 este reprezentat arborele de ocupare a regiunii.

Arborele se creează pornind cu o regiune pătrată în plan, reprezentând întrega zonă care se modelează şi care este reprezentată prin nodul rădăcină al arborelui cuaternar. În cazul spaţiului tridimensional, acestă regiune este un cub. Fiecare regiune, începând cu regiunea iniţială, se subdivide în patru subregiuni, reprezentate ca patru noduri fii în arbore. În Fig. 2.13(b) se arată ordinea de numerotare a nodurilor fii obţinuţi prin divizarea unei subregiuni.

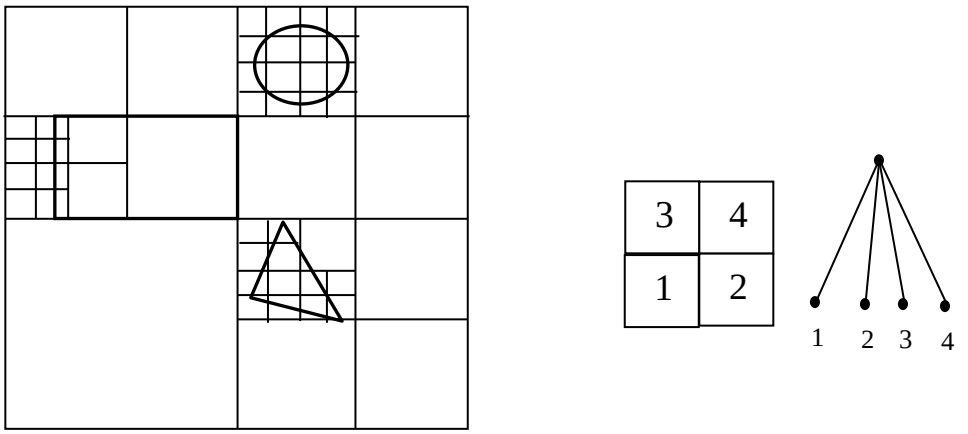


Fig. 2.13 (a) Ocuparea unei regiuni plane.
(b) Ordinea de numerotare a nodurilor fii.

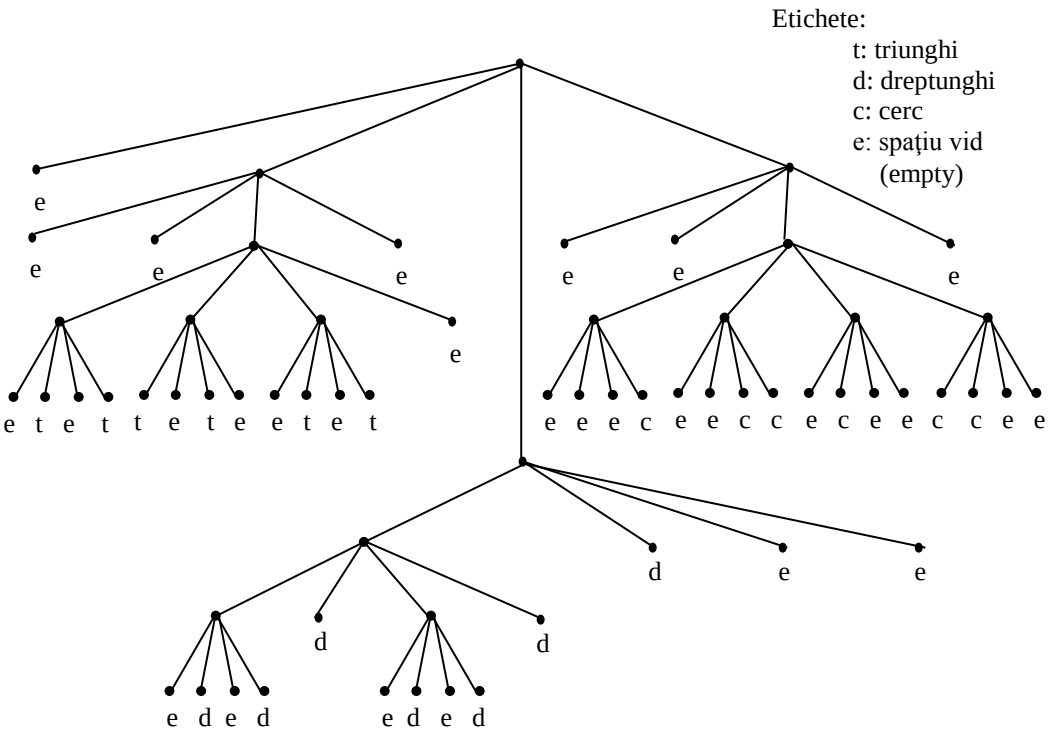


Fig. 2.14 Arborele de reprezentare a ocupării spaţiale.

În spaţiul tridimensional o regiune este divizată în opt subregiuni, reprezentate prin opt noduri fii; de aici provine şi numele octree acestui mod de reprezentare.

Subregiunile sunt divizate recursiv până se întâlneşte una din următoarele situaţii:

- Subregiunea este ocupată în întregime de un singur obiect sau nu este ocupată de nici un obiect. Unei astfel de regiuni i se atribuie eticheta cu numele obiectului respectiv, sau eticheta de spaţiu vid (e).
- Subregiunea are dimensiunea minimă admisibilă în reprezentarea respectivă (un pixel). În acest caz, subregiunea primeşte eticheta obiectului (sau a spaţiului vid) care ocupă cea mai mare parte din subregiunea respectivă.

În arborele de reprezentare a ocupării spaţiului tridimensional, dimensiunea minimă până la care se divizează subregiunile este dimensiunea unui voxel. În arborele de reprezentare sunt două tipuri de noduri frunză: noduri care au o etichetă a unui obiect sau noduri care au eticheta spaţiului vid. Compactarea volumului de date în această reprezentare faţă de reprezentarea prin etichetarea fiecărui element (pixel sau voxel), provine din faptul că se poate ca o întreagă subregiune (mai mare decât un voxel) primeşte o singură etichetă, dacă este nu este ocupată deloc sau este ocupată de un singur obiect. În Fig 2.13 (a), regiunea 1 din prima subdivizune nu mai este subdivizată mai departe deoarece este în întregime vidă şi primeşte eticheta (e).

O variantă de reprezentare a ocupării spaţiale prin arbori octali foloseşte ca element spaţial un tetraedru regulat, care este subdivizat în opt tetraedre, până la rezoluţia dorită sau până când este ocupat de un singur obiect din spaţiu. Avantajul folosirii arborilor octali bazaţi pe tetraedre este că se poate obţine direct o reprezentare echivalentă a suprafeţei triangularizate de aproximare a obiectului.

Redarea imaginii obiectelor modelate prin divizare spaţială se face fie prin conversie în reprezentarea prin suprafaţa de frontieră, fie direct, prin *redare volumetrică*, adică redarea tuturor elementelor de volum (voxeli).

3

Transformări geometrice

În procesul de redare a obiectelor tridimensionale se definesc mai multe sisteme de referinţă, care permit descrierea operaţiilor de transformări succesive ale obiectelor, pornind de la modelul acestora, până la imaginea bidimensională pe suportul de reprezentare.

Aceste sisteme de referinţă vor fi precizate pe parcursul expunerii; ceea ce interesează în acest moment este faptul că în vizualizarea scenelor virtuale sunt necesare (şi folosite) mai multe sisteme de referinţă în care sunt reprezentate obiectele. Modificarea unui obiect într-un sistem de referinţă dat, sau trecerea de la un sistem de referinţă la altul se realizează prin diferite transformări grafice. Dintre transformările grafice folosite, unele modifică forma şi poziţia obiectelor în spaţiu, fiind numite transformări geometrice, altele sunt transformări de conversie între diferite modalităţi de reprezentare a obiectelor (transformarea de rastru).

Transformări geometrice în spaţiu

Obiectele scenei virtuale pot fi modificate sau manevrate în spaţiul tridimensional, folosind diferite transformări geometrice. Dintre acestea, cele mai importante sunt translaţia, care modifică localizarea obiectului, rotaţia, care modifică orientarea, şi scalarea, care modifică dimensiunea obiectului. Aceste transformări sunt denumite transformări geometrice primitive.

3.1.1 Transformări geometrice primitive

Translaţia este transformarea prin care toate punctele se deplasează în aceeaşi direcţie şi cu aceeaşi distanţă între punct şi transformatul său. Translaţia se poate descrie printr-un vector de translaţie **T**, având componentele t_x, t_y, t_z pe cele trei axe de coordonate; un punct $P(x, y, z)$ se transformă în punctul $P'(x', y', z')$ astfel:

$$x' = x + t_x$$
$$y' = y + t_y$$
$$z' = z + t_z$$

În notaţie matriceală, transformarea prin translaţie cu vectorul de translaţie **T**, având componentele t_x, t_y, t_z pe cele trei axe de coordonate a unui punct $P(x, y, z)$ în punctul $P'(x', y', z')$ se exprimă printr-o însumare de matrice:

$$\mathbf{P' = P + T}, \text{ unde } \mathbf{T} = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} ; \text{ deci: } \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}.$$

Scalarea modifică coordonatele tuturor punctelor unui obiect folosind factorii de scală s_x, s_y, respectiv s_z pe cele trei axe de coordonate. În această transformare de scalare, numită scalare faţă de origine, un punct $P(x, y, z)$ se transformă în punctul $P'(x', y', z')$, unde:

$$x' = x \cdot s_x$$
$$y' = y \cdot s_y$$
$$z' = z \cdot s_z$$

Pentru scrierea sub formă matriceală a acestor relaţii de transformare, se defineşte matricea de scalare **S** de dimensiune 3x3 astfel:

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

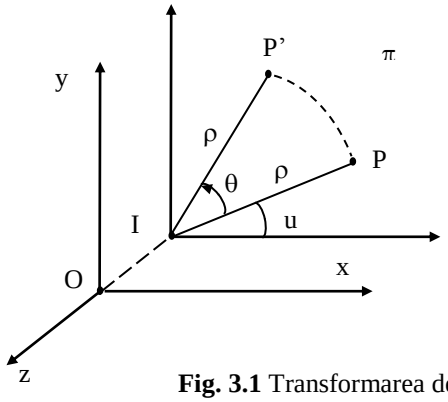
Rezultă relaţia de transformare de scalare în notaţie matriceală:

$$\mathbf{P' = S\,P}, \text{ adică } \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Rotaţia cu un unghi θ în raport cu o axă D este o transformare prin care orice punct P care nu este situat pe dreapta d se transformă într-un punct P’ astfel încât P şi P’ sunt situate într-un plan perpendicular π pe dreapta D, unghiul PIP’este egal cu θ, iar modulele |IP| şi |IP’| sunt egale (punctul I este intersecţia dintre dreapta D şi planul π). Prin această transformare toate punctele dreptei d sunt fixe şi numai ele sunt puncte invariante ale transformării. Transformarea de rotaţie în raport cu o axă oarecare se

descompune o într-o succesiune de maximum trei transformări de rotaţie în raport cu axele de coordonate ale sistemului de referinţă.

Rotaţia în raport cu axa z cu un unghi θ transformă un punct $P(x, y, z)$ în punctul $P'(x', y', z')$, ambele aflate în planul π peroendicular pe axa z. Pentru deducerea relaţiilor de transformare se exprimă coordonatele punctelor P şi P' în acest plan în coordonate polare (Fig. 3.1).



$$P \begin{cases} x = \rho \cos u \\ y = \rho \sin u \end{cases}$$

$$P' \begin{cases} x' = \rho \cos(u + \theta) \\ y' = \rho \sin(u + \theta) \end{cases}$$

Fig. 3.1 Transformarea de rotaţie în raport cu axa z.

Se dezvoltă cosinusul şi sinusul sumei de unghiuri şi se obţin relaţiile:

$$x' = \rho(\cos u \cos\theta - \sin u \sin\theta) = x\cos\theta - y\sin\theta$$

$$y' = \rho(\sin u \cos\theta + \sin\theta \cos u) = x\sin\theta + y\cos\theta$$

Această transformare se poate scrie sub formă matriceală dacă se defineşte matricea de rotaţie $\mathbf{R_Z}(\theta)$ de dimensiune 3x3 astfel:

$$\mathbf{R_Z}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rezultă relaţiile de transformare de rotaţie a unui punct în raport cu axa z cu un unghi θ scrise sub formă matriceală:

$$\mathbf{P}' = \mathbf{R_Z P}, \quad \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Pentru rotaţiile elementare ale unui punct în raport cu axele x şi y ale sistemului de referinţă se urmăreşte un raţionament asemănător şi se deduc relaţiile de transformare corespunzătoare.

Rotaţiile în raport cu axele de coordonate ale sistemului de referinţă sunt denumite tangaj (pitch), giraţie (yaw, heading) şi, respectiv ruliu (roll). Această asignare depinde de convenţiile de definire a sistemului de referinţă universal. Pentru convenţia definită mai sus, tangajul este o rotaţie în raport cu axa x, giraţia este o rotaţie în raport cu axa y, iar ruliul este o rotaţie în raport cu axa z (Fig. 3.2).

Aceste denumiri au originea în modul în care sunt definite mişcările unui avion poziţionat în spaţiu cu axa longitudinală orientată către z pozitiv: tangajul este rotaţia într-un plan vertical, care înclină botul avionului; giraţia este o mişcare într-un plan orizontal, care schimbă direcţia axei avionului, iar ruliul este rotaţia într-un plan vertical, care înclină aripile avionului.

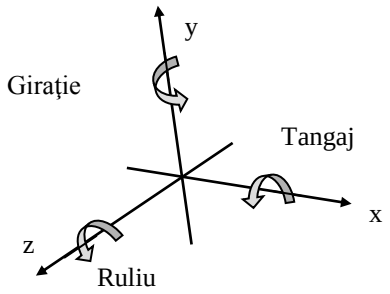


Fig. 3.2 Rotaţiile în raport cu axele sistemului de referinţă.

Semnul rotaţiilor în raport cu axele de coordonate se alege, prin convenţie, după regula mâinii drepte sau după regula mâinii stângi. Dacă se cuprinde axa corespunzătoare cu patru degete ale mâinii (drepte, respectiv, stângi) astfel ca degetul mare să fie îndreptat în sensul pozitiv al axei, direcţia celor patru degete indică sensul pozitiv al rotaţiei. În această lucrare se adoptă regula mâinii drepte pentru sensul de rotaţe în raport cu axele de coordonate (Fig. 3.2).

S-au obţinut relaţiile pentru transformările spaţiale elementare (primitive), translaţia, scalarea faţă de originea sistemului de coordonate şi rotaţia în raport cu axele sistemului de coordonate. Tratarea într-un mod unitar a acestor transformări se poate face prin creşterea dimensionalităţii sistemului de coordonate Cartezian, definindu-se un sistem de coordoonate cu 4 dimensinui, numit sistem de coordonate omogene.

3.1.2 Sistemul de coordonate omogene

Se observă că transformările de scalare şi de rotaţie se reprezintă prin înmulţiri de matrice, iar translaţia se reprezintă prin însumare de matrice. Reprezentarea unitară şi combinarea transformărilor geometrice spaţiale se poate face într-un sistem de coordonate cu patru dimensiuni, numit sistem de coordonate omogene. În sistemul de coordonate omogene, un punct $P(x, y, z)$ se reprezintă ca $P(X, Y, Z, w)$ pentru orice factor de scară $w \neq 0$.

În general, un punct $P(x, y, z)$ în sistemul de coordonate Cartezian se transformă în sistemul de coordonate omogene prin alegerea $w = 1$, deci are coordonatele omogene $P(x, y, z, 1)$. Pentru transformarea inversă, se calculează coordonatele carteziene ale unui punct $P(X, Y, Z, w)$ reprezentat în sistemul de coordonate omegene printr-o împărţire cu factorul de scară w astfel:

$$x = X / w$$
$$y = Y / w \qquad (3.1)$$
$$z = Z / w$$

În coordonate omogene, două puncte $P_1(X_1, Y_1, Z_1, w_1)$ şi $P_2(X_2, Y_2, Y_2, w_2)$ sunt egale dacă :

$$X_1 / w_1 = X_2 / w_2,\ Y_1 / w_1 = Y_2 / w_2 \text{ si } Z_1 / w_1 = Z_2 / w_2$$

În sistemul de coordonate omogene, toate matricele de transformări sunt de dimensiune 4x4 şi toate operaţiile de transformări geometrice se pot exprima prin produse de matrice.

Matricea de translaţie în coordonate omogene se defineşte astfel:

$$\mathbf{T}(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.2)$$

Transformarea de translaţie a unui punct $P(X, Y, Z, w)$ reprezentat în coordonate omogene, în punctul $P'(X', Y', Z', w')$ se exprimă ca un produs de matrice:

$$\mathbf{P'} = \mathbf{T}\,\mathbf{P}$$

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix}, \text{ rezultă: } \begin{cases} X' = X + wt_x \\ Y' = Y + wt_y \\ Z' = Z + wt_z \\ w' = w \end{cases} \qquad (3.3)$$

Teoretic, pentru a se obţine coordonatele cartezine ale punctului transformat P', trebuie să se execute operaţia de împărţire cu factorul de scară w' pentru fiecare coordonată:

$$x' = X' / w'$$
$$y' = Y' / w'$$
$$z' = Z' / w'$$

Dar, dacă se alege w =1, rezultă w'=1 şi împărţirea nu mai este necesară. În general, transformările geometrice primitive conservă valoarea factorului de scară şi, dacă se alege w =1, împărţirea cu w' (numită împărţirea omogenă) nu este necesară .

Matricea de scalare faţă de origine reprezentată în sistemul de coordonate omogene are expresia:

$$\mathbf{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

Transformarea de scalare a unui punct reprezentat în coordonate omogene $P(X, Y, Z, w)$, în punctul $P'(X', Y', Z', w')$ este dată de relaţiile:

$$\mathbf{P'} = \mathbf{SP}$$

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ w' \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix}, \text{ rezultă: } \begin{cases} X' = s_x X \\ Y' = s_y Y \\ Z' = s_z Z \\ w' = w \end{cases} \tag{3.5}$$

Dacă factorii de scalare sunt egali ($s_x = s_y = s_z$), scalarea se numeşte uniformă, şi păstrează forma obiectului. Dacă factorii de scalare diferă, obiectul este deformat, iar scalarea se numeşte neuniformă.

Transformările de rotaţie în raport cu axele sistemului de referinţă exprimate în coordonate omogene sunt următoarele:

Rotaţia în raport cu axa x (tangaj) cu un unghi θ :

$$\mathbf{R_X}(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

În această transformare, axa x rămâne nemodificată, iar toate celelalte puncte din spaţiu se transformă prin înmulţire ca matricea $\mathbf{R_X}(\theta)$:

$$\mathbf{P'} = \mathbf{R_X P}$$

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ w \end{bmatrix}, \text{ rezultă } \begin{cases} X' = X \\ Y' = Y\cos\theta - Z\sin\theta \\ Z' = Y\sin\theta + Z\cos\theta \\ w' = w \end{cases} \tag{3.7}$$

Rotaţia în raport cu axa y (giraţie) cu un unghi χ :

$$\mathbf{R_Y}(\chi)=\begin{bmatrix}\cos\chi & 0 & \sin\chi & 0\\ 0 & 1 & 0 & 0\\ -\sin\chi & 0 & \cos\chi & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{3.8}$$

În această transformare, axa y rămâne nemodificată, iar toate celelalte puncte din spaţiu se transformă prin înmulţire ca matricea $\mathbf{R_Y}(\chi)$:

$$\mathbf{P'}=\mathbf{R_Y}\mathbf{P}$$

$$\begin{bmatrix}X'\\ Y'\\ Z'\\ w'\end{bmatrix}=\begin{bmatrix}\cos\chi & 0 & \sin\chi & 0\\ 0 & 1 & 0 & 0\\ -\sin\chi & 0 & \cos\chi & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}X\\ Y\\ Z\\ w\end{bmatrix}, \text{ rezultă: } \begin{cases}X'=X\cos\chi+Z\sin\chi\\ Y'=Y\\ Z'=-X\sin\chi+Z\cos\chi\\ w'=w\end{cases} \tag{3.9}$$

Rotaţia în raport cu axa z (ruliu), cu un unghi ρ :

$$\mathbf{R_Z}(\rho)=\begin{bmatrix}\cos\rho & -\sin\rho & 0 & 0\\ \sin\rho & \cos\rho & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{3.10}$$

În această transformare, axa z rămâne nemodificată, iar toate celelalte puncte din spaţiu se transformă prin înmulţire ca matricea $\mathbf{R_Z}(\rho)$:

$$\mathbf{P'}=\mathbf{R_Z}\mathbf{P}$$

$$\begin{bmatrix}X'\\ Y'\\ Z'\\ w'\end{bmatrix}=\begin{bmatrix}\cos\rho & -\sin\rho & 0 & 0\\ \sin\rho & \cos\rho & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}X\\ Y\\ Z\\ w\end{bmatrix}, \text{ rezultă: } \begin{cases}X'=X\cos\rho-Y\sin\rho\\ Y'=X\sin\rho+Y\cos\rho\\ Z'=Z\\ w'=w\end{cases} \tag{3.11}$$

Toate matricele de transformare de rotaţie sunt matrice ortogonale şi ortonormate. Transformările geometrice elementare sunt transformări liniare, prin care liniile drepte şi suprafeţele plane sunt transformate în linii dreapte, respectiv suprafeţe plane. Din această cauză, pentru transformarea unui obiect tridimensional, este suficient să se transforme toate vârfurile acestuia şi să se păstreze relaţiile topologice între vârfurile transformate, aceleaşi cu cele între vârfurile iniţiale.

Transformările mai complexe ale obiectelor în spaţiu se pot defini prin compunerea mai multor transformări primitive.

3.1.3 Compunerea transformărilor geometrice

Compunerea mai multor tranformări elementare pentru obţinerea unei transformări complexe se obţine prin executarea succesivă a produsului fiecăriei matrice de transformare cu matricea de reprezentare a punctului iniţial sau rezultat dintr-o transformare precedentă. De exemplu, în Fig. 3.3 este prezentată o transformare compusă a unui obiect.

Obiectul iniţial este un cub cu latura de două unităţi, amplasat cu centrul său în centrul sistemului de referinţă şi laturile orientate în direcţiile axelor de coordonate. După o scalare cu factorii de scară 2, 2, 2, o rotaţie cu un unghi de 30 grade în raport cu axa z şi o translaţie cu un vector de translaţie cu componente 8,0,0, se obţine un nou obiect cub, definit în acelaşi sistem de referinţă, dar cu alte dimensiuni şi localizare. Reprezentarea din Fig. 3.3 conţine şi o transformare de proiecţie perspectivă, pentru percepţia adâncimii (a distanţei faţă de observator a obiectelor), care va fi explicată ulterior. Una din feţele cubului (cea mai apropiată de observator) este desenată ca suprafaţă de culoare gri; celelalte feţe ale cubului sunt reprezentate numai prin muchiile lor (reprezentare numită "cadru de sârmă", *wireframe*).

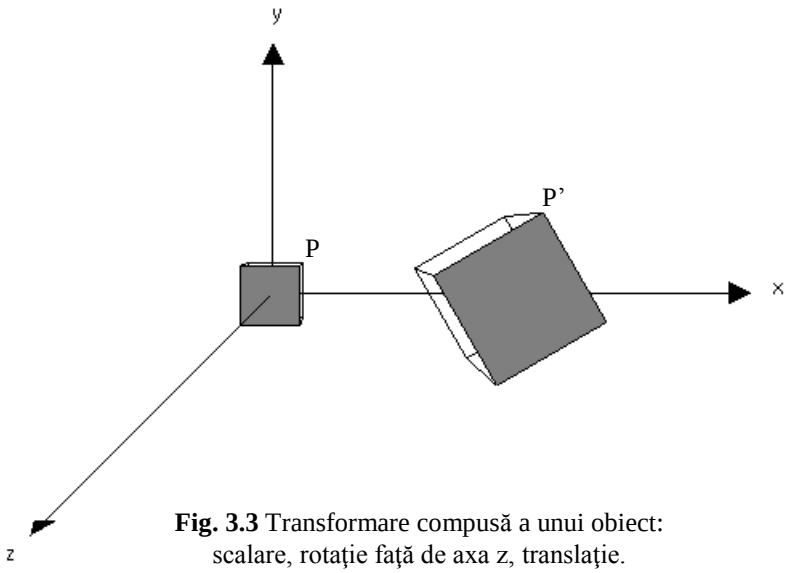


Fig. 3.3 Transformare compusă a unui obiect: scalare, rotaţie faţă de axa z, translaţie.

Transformarea efectuată asupra cubului din figura de mai sus se obţine prin aplicarea succesivă a trei transformări geometrice elementare (scalare, rotaţie şi translaţie) asupra fiecărui punct (vârf) al cubului. Pentru un vârf al cubului, reprezentat prin matricea coloană **P**, succesiunea de transformări este:

Scalarea:	$\mathbf{P_1 = S\,P}$
Rotaţia în raport cu axa z:	$\mathbf{P_2 = R_Z\,P_1 = R_Z(S\,P)}$
Translaţia:	$\mathbf{P' = T\,P_2 = T(R_Z(S\,P)) = (TR_ZS)\,P}$

Nu este necesar să fie executate pe rând operaţiile de înmulţire cu matricele de transformare **S**, $\mathbf{R_Z}$, şi **T**, ci se poate calcula o matrice compusă a transformării **M**, care se aplică apoi fiecărui punct **P** al obiectului:

$\mathbf{P'} = \mathbf{T}\mathbf{R_Z}\mathbf{S}\mathbf{P} = \mathbf{M}\mathbf{P}$, unde $\mathbf{M} = \mathbf{T}\mathbf{R_Z}\mathbf{S}$

O transformare compusă se poate deci defini printr-o matrice de transformare **M** care este un produs (compunere) de matrice de transformări geometrice elementare.

În exemplul dat, se calculează matricea de transformare **M** astfel:

$$\mathbf{M} = \mathbf{T}\mathbf{R_z}\mathbf{S} = \begin{bmatrix} 1 & 0 & 0 & 8 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.866 & -0.5 & 0 & 0 \\ 0.5 & 0.866 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M} = \begin{bmatrix} 1.732 & -1 & 0 & 8 \\ 1 & 1{,}732 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Unul din vârfurile cubului, vârful P(1,1,1,1) se transformă în P'(8.732, 3.732,1,1):

$$\mathbf{P'} = \mathbf{M}\mathbf{P} = \begin{bmatrix} 1.732 & -1 & 0 & 8 \\ 1 & 1.732 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 8.732 \\ 3.732 \\ 2 \\ 1 \end{bmatrix}$$

Ordinea de compunere a matricelor de transformare este definitorie pentru rezultatul transformării, dat fiind că produsul matricelor nu este comutativ.

Se poate verifica pe exemplul dat, prin inversarea ordinii transformării de rotaţie cu translaţia (Fig. 3.4).

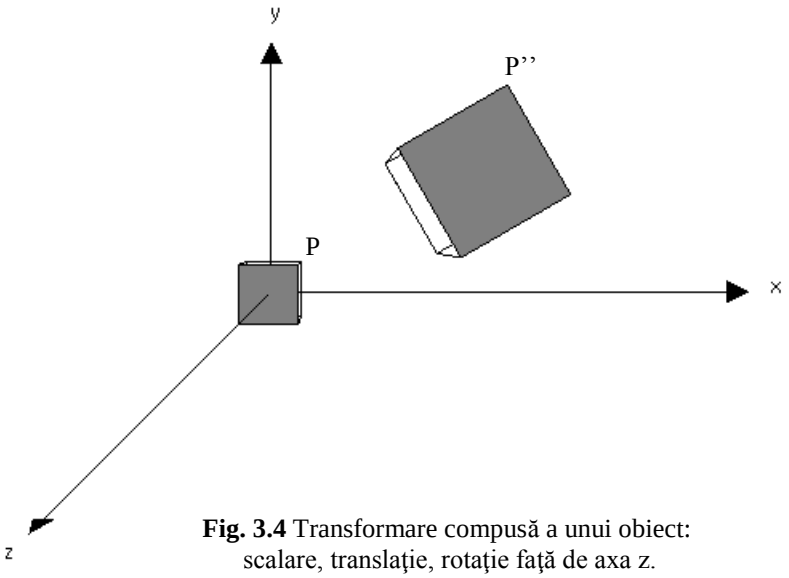


Fig. 3.4 Transformare compusă a unui obiect:
scalare, translaţie, rotaţie faţă de axa z.

Se obţine o matrice de transformare **M''**diferită de **M** şi punctul transformat corespunzător P'', diferit de P':

$$\mathbf{M''} = \mathbf{R_z TS} = \begin{bmatrix} 1.732 & -1 & 0 & 6.928 \\ 1 & 1.732 & 0 & 4.330 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{P''} = \mathbf{M''P} = \begin{bmatrix} 7.560 \\ 7.06 \\ 2 \\ 1 \end{bmatrix}$$

Convenţia de reprezentare a punctelor în spaţiu prin matrice coloană impune ordinea de înmulţire numită postmultiplicare (sau multiplicare la dreapta) a matricelor de transformare: se înmulţeşte matricea de transformare curentă cu matricea transformării următoare. Convenţia de terminologie (postmultiplicare sau multiplicare la dreapta) semnifică faptul că o nouă transformare se concatenează ca factor dreapta al produsului de matrice. Este important de reţinut faptul că ordinea de aplicare a transformărilor este de la dreapta la stânga din succesiunea de matrice ale unei compuneri. Mai precis, dacă un punct se transformă prin aplicarea succesivă a transformărilor definite prin matricele $\mathbf{M_1}$, $\mathbf{M_2}$,…., $\mathbf{M_n}$, matricea compusă de transformare este:

$$\mathbf{M} = \mathbf{M_n} \cdots \mathbf{M_2 M_1} \tag{3.12}$$

Se poate verifica că, dacă se adoptă convenţia de reprezentare a unui punct în spaţiul tridimensional printr-o matrice linie, atunci transformarea unui punct se obţine prin înmulţirea vectorului de poziţie al punctului cu matricea de transformare (premultiplicare sau multiplicare la stânga). În această situaţie, ordinea în care se aplică matricele compunente ale unei transformări compuse este de la stânga la dreapta:

$$\mathbf{P'} = \mathbf{PS}, \quad \text{unde } \mathbf{S} = \mathbf{S_1 S_2} \cdots \mathbf{S_n} \tag{3.13}$$

Punctul P este transformat în ordinea $\mathbf{S_1}$, $\mathbf{S_2}$,….$\mathbf{S_n}$. Acesta este un mod mai natural de a urmări secvenţele de transformări geometrice şi de aceea, în unele lucrări este adoptată această convenţie. Nu există alte diferenţe între cele două convenţii (nici de simplificare a calculelor, nici de eficienţă).

Se poate demonstra cu uşurinţă că orice matrice de transformare elementară în convenţia de reprezentare prin matrice linie a unui punct în spaţiul tridimensional este transpusa matricei corespunzătoare de transformare definită în convenţia de reprezentare prin matrice coloană a punctului. De exemplu, translaţia unui punct P(x,z,y) cu valorile t_x, t_y, t_z se exprimă astfel:

$$\begin{bmatrix} X' & Y' & Z' & w' \end{bmatrix} = \begin{bmatrix} X & Y & Z & w \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

În această lucrare se foloseşte convenţia de reprezentare prin matrice coloană a punctelor din spaţiu, dat fiind că aceasta este convenţia din biblioteca grafică OpenGL, care este folosită pentru exemplificarea operaţiilor grafice prezentate.

Un alt exemplu de transformare compusă este transformarea de rotaţie completă specificată prin trei rotaţii faţă de axele sistemului de coordonate. Cea mai obişnuită convenţie pentru ordinea de specificare a rotaţiilor este: ruliu cu unghiul ρ (după axa z), tangaj cu unghiul θ (după axa x) şi giraţie cu unghiul χ (faţă de axa y). În această situaţie matricea de rotaţie totală **R** are expresia:

$$\mathbf{R} = \mathbf{R_Y}(\chi)\mathbf{R_X}(\theta)\mathbf{R_Z}(\rho) = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.14)$$

Matricea **R** rezultată prin compunerea (înmulţirea) mai multor matrice de rotaţie este de asemenea ortogonală şi ortonormată.

O transformare complexă a unui obiect prin combinarea mai multor transformări elementare (scalări, rotaţii, translaţii) se poate exprima printr-o matrice de transformare **M** care are forma generală:

$$\mathbf{M} = \begin{bmatrix} r'_{11} & r'_{12} & r'_{13} & t_x \\ r'_{21} & r'_{22} & r'_{23} & t_y \\ r'_{31} & r'_{32} & r'_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.15)$$

Submatricea stânga-sus **R'** de dimensiune 3x3 exprimă transformarea de rotaţie şi scalare totală, iar submatricea coloană dreapta **T** exprimă transformarea de translaţie totală. Matricea **R'** este o matrice ortogonală, dar poate să nu fie şi ortonormată.

Două proprietăţi importante ale transformărilor geometrice compuse trebuie să fie remarcate şi reţinute.

Prima proprietate este că o transformare geometrică compusă reprezentată printr-o matrice de forma 3.15 conservă valoarea coordonatei w a unui punct (acest lucru este evident). Această proprietate este folosită în implementarea operaţiilor grafice, prin amânarea împărţirii omogene (cu w) după ce s-au selectat numai obiectele vizibile (după operaţia de decupare).

A doua proprietate este proprietatea de liniaritate a transformărilor geometrice compuse, prin care liniile drepte şi suprafeţele plane sunt transformate în linii dreapte, respectiv suprafeţe plane. De aceea, la fel ca şi în cazul transformărilor geometrice

elementare, pentru transformarea unui obiect tridimensional, este suficient să se transforme toate vârfurile acestuia şi să se păstreze relaţiile topologice între vârfurile transformate.

3.1.4 Transformări inverse

Fiind dată o transformare a unui punct P într-un punct P' definită printr-o matrice de transformare **M**, transformarea inversă, de la punctul P' la punctul P se obţine prin înmulţirea cu matricea inversă, $\mathbf{M}^{-1}$:

$$\mathbf{M}\mathbf{M}^{-1} = \mathbf{I},\ \mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{3.16}$$

unde **I** este matricea identitate.

Dat fiind că, în general, matricea de transformare **M** se obţine printr-un produs de matrice de transformări elementare, matricea inversă $\mathbf{M}^{-1}$ se calculează prin produsul în ordine inversă a inverselor matricelor elementare componente:

$$\begin{aligned} &\mathbf{M} = \mathbf{M_n} \cdots \mathbf{M_2}\mathbf{M_1} \\ &\mathbf{M}^{-1} = \mathbf{M_1}^{-1}\mathbf{M_2}^{-1} \cdots \mathbf{M_n}^{-1} \\ &\mathbf{M}\mathbf{M}^{-1} = \mathbf{M_n} \cdots \mathbf{M_2}\mathbf{M_1}\mathbf{M_1}^{-1}\mathbf{M_2}^{-1} \cdots \mathbf{M_n}^{-1} = \mathbf{I} \end{aligned} \tag{3.17}$$

Relaţia 3.17 se demonstrează imediat, prin gruparea factorilor începînd cu $\mathbf{M_1}\mathbf{M_1}^{-1} = \mathbf{I}$.

Toate matricele de transformări elementare sunt matrice inversabile şi au următoarele expresii:

$$[\mathbf{T}(t_x, t_y, t_z)]^{-1} = \mathbf{T}(-t_x, -t_y, -t_z) = \begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.18}$$

$$[\mathbf{S}(s_x, s_y, s_z)]^{-1} = \mathbf{S}(1/s_x, 1/s_y, 1/s_z) = \begin{bmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.19}$$

$$[\mathbf{R_x}(\theta)]^{-1} = \mathbf{R_x}(-\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.20}$$

$$[\mathbf{R_y}(\chi)]^{-1} = \mathbf{R_y}(-\chi) = \begin{bmatrix} \cos\chi & 0 & -\sin\chi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\chi & 0 & \cos\chi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.21}$$

$$[\mathbf{R}_z(\rho)]^{-1} = \mathbf{R_z}(-\rho) = \begin{bmatrix} \cos\rho & \sin\rho & 0 & 0 \\ -\sin\rho & \cos\rho & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.22}$$

$$\mathbf{R}^{-1} = [\mathbf{R_y}(\chi)\mathbf{R_x}(\theta)\mathbf{R_z}(\rho)]^{-1} = \mathbf{R_z}(-\rho)\mathbf{R_x}(-\theta)\mathbf{R_y}(-\chi) \tag{3.23}$$

Datorită faptului că matricea de rotaţie este ortogonală şi ortonormată, inversa unei matrice de rotaţieeste egală cu transpusa acesteia.

3.1.5 Transformarea sistemelor de referinţă

Interpretarea relaţiilor de transformări geometrice prezentate până acum a fost aceea de manevrare şi modificare a obiectelor într-un sistem de referinţă dat: obiectele sunt reprezentate într-un anumit sistem de referinţă prin coordonatele unei mulţimi de puncte ale acestora (vârfuri) şi ele pot fi deplasate, reorientate sau redimensionate prin aplicarea transformărilor geometrice corespunzătoare.

O altă interpretare care se poate da operaţiilor de transformări geometrice este aceea de schimbare a sistemului de referinţă.

Se consideră sistemul de referinţă Oxyz şi un nou sistem de referinţă O'x'y'z', a cărui origine O' este determinată în sistemul Oxyz prin coordonatele x_0,y_0,z_0 ale centrului O'. Sistemul de referinţă Oxyz este definit de versorii (vectori unitate) **i, j, k,** iar sistemul de referinţă O'x'y'z' de versorii **i', j', k'.**

Axa O'x' are cosinuşii directori c_{11}, c_{12}, c_{13} faţă de sistemul de referinţă Oxyz; axa O'y' are cosinuşii directori c_{21}, c_{22}, c_{23} faţă de sistemul de referinţă Oxyz; axa O'z' are cosinuşii directori c_{31}, c_{32}, c_{33} faţă de sistemul de referinţă Oxyz.

Matricea de transformare care descrie poziţionarea sistemului O'x'y'z' relativ la sistemul de referinţă Oxyz este:

$$\mathbf{M} = \begin{bmatrix} c_{11} & c_{21} & c_{31} & x_0 \\ c_{12} & r_{22} & c_{32} & y_0 \\ c_{13} & c_{23} & c_{33} & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.24}$$

Fie un punct P definit prin coordonatele sale x,y,z în sistemul de referinţă Oxyz. Se demonstrează [Drag57], [Sab81], că în sistemul de referinţă O'x'y'z' acest punct (notat P') are coordonatele x',y',z', care se obţin înmulţirea matricei $\mathbf{M^{-1}}$ (care este inversa matricei **M** care defineşte poziţionarea sistemului O'x'y'z' în sistemul Oxyz) cu matricea **P** de reprezentare a punctului P în sistemul de coordonate iniţial:

$$\mathbf{P'} = \mathbf{M}^{-1}\mathbf{P} \tag{3.25}$$

Transformarea inversă, a unui punct P'(x',y',z') din sistemul de referinţă O'x'y'z' în punctul P(x,y,z) în sistemul de referinţă Oxyz se obţine prin înmulţire cu matricea de transformare **M**:

$$\mathbf{P} = \mathbf{M}\mathbf{P'} \tag{3.26}$$

Se poate urmări cu mai multă uşurinţă această modalitate de transformare într-un caz simplu. Se consideră un sistem de referinţă Oxyz şi un alt sistem de referinţă O'x'y'z' care are originea O'(x_0,y_0,z_0) şi aceeaşi orientare a axelor de coordonate ca şi sistemul Oxyz.

Poziţionarea sistemului de referinţă O'x'y'z' relativ la sistemul Oxyz este definită de matricea de transformare **M**; poziţionarea sistemului de referinţă Oxyz relativ la sistemul O'x'y'z' este definită de matricea de transformare inversă $\mathbf{M}^{-1}$:

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & x_0 \\ 0 & 1 & 0 & y_0 \\ 0 & 0 & 1 & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{M}^{-1} = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Submatricea de rotaţie componentă a matricei **M** este matricea unitate, dat fiind că sistemul de referinţă O'x'y'z' are axele cu aceeaşi orientare ca şi sistemul de referinţă. Într-adevăr, cosinuşii directori ai axelor O'x',O'y',O'z' faţă de axele sistemului de referinţă Oxyz sunt (1,0,0), (0,1,0), (0,0,1) şi matricea de rotaţie este matricea unitate.

Un punct oarecare P(x,y,z) în sistemul de referinţă Oxyz, se transformă în punctul P'(x',y',z') în sistemul de referinţă O'x'y'z' printr-o translaţie cu $-x_0$,$-y_0$,$-z_0$, deci :

$$\mathbf{P'} = \mathbf{M}^{-1}\mathbf{P}; \quad \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Particularizarea pentru O'(4,2,0) şi P(6,3,0) este reprezentată în Fig. 3.5. Punctul P'(2,1,0) este transformatul punctului P în sistemul de referinţă O'x'y'z'.

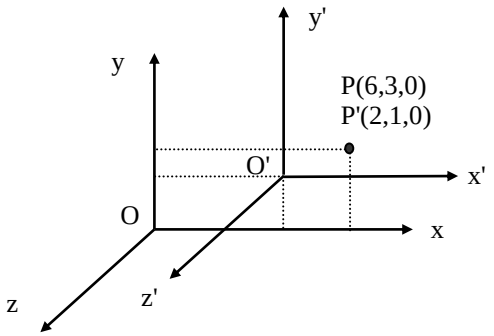


Fig.3.5 Transformarea sistemelor de referinţă.

În concluzie, aplicarea unei transformări definită printr-o matrice **M**, asupra unei mulţimi de puncte definite într-un sistem de referinţă Oxyz, poate fi interpretată în mai multe moduri:

(a) Fiecare punct este modificat şi capătă o nouă poziţionare în sistemul de referinţă Oxyz, conform cu matricea de transformare **M**.
(b) Fiecare punct este transformat din sistemul de referinţă iniţial Oxyz într-un nou sistem de referinţă, O'x'y'z', a cărui poziţie şi orientare relativ la sistemul de referinţă Oxyz este descrisă de matricea $\mathbf{M}^{-1}$.
(c) Fiecare punct este transformat din sistemul de referinţă iniţial Oxyz într-un nou sistem de referinţă, O'x'y'z'. Poziţia şi orientarea sistemului Oxyz relativ la sistemul de referinţă O'x'y'z' este descrisă de matricea **M**.

Şi încă o precizare: un sistem de referinţă nu este "materializat" în nici un fel într-un sistem grafic (în software sau hardware), ci este o convenţie cunoscută de programator sau utilizator. Trecerea de la un sistem de referinţă la altul se efectuează prin transformări aplicate punctelor (obiectelor): modificarea coordonatelor acestora le transferă dintr-un sistem de referinţă în altul. În acest sens, orice transformare geometrică poate fi considerată ca o schimbare a sistemului de referinţă. Dar, după cum se va observa în continuare, transformările geometrice sunt uneori considerate ca modificări ale obiectelor, iar în alte situaţii, ca schimbare a sistemului de referinţă. Este normal ca un începător în practica programării sistemelor grafice să pună o justificată întrebare: cum se poate şti care este interpretarea corectă a unei transformări? Răspunsul este că ambele interpretări sunt corecte, dar se alege aceea care permite urmărirea cea mai directă a unui raţionament pentru calculul unor transformări complexe. Nu există "reţete" unice şi sigure care să poată fi aplicate fără greşeală în orice situaţie, deci experienţa de proiectare îşi spune întotdeauna cuvântul.

În dezvoltarea sistemelor de vizualizare se folosesc, totuşi, câteva sisteme de referinţă bine definite, care permit specificarea cea mai simplă şi eficientă a operaţiilor de redare a obiectelor (scenelor). Unul dintre acestea, sistemul de referinţă universal, a fost deja introdus. Alte sisteme de referinţă folosite în grafica tridimensională vor fi definite pe parcursul lucrării.

Sistemele de referinţă intermediare, care apar în descrierea unor transformări complexe, sunt interpretări ale operaţiilor de transformări geometrice care permit urmărirea unui raţionament de calcul. Exemplul următor evidenţiază acest aspect.

ν **Exemplul 3.2**

Se consideră sistemul de coordonate Oxyz şi o matrice de transformare $\mathbf{R}_X(\pi/2)$ care realizează o rotaţie cu unghiul $\pi/2$ în raport cu axa x:

$$\mathbf{R_X}(\pi/2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Această transformare poate fi interpretată ca o transformare prin rotaţie cu 90° în raport cu axa x a fiecărui punct P(x,y,z), în punctul transformat P'(x',y',z'). În Fig.3.6 (a) este considerat punctul P(0,1,1), care se transformă în punctul P'(0,-1,1).

În Fig. 3.6(b) este reprezentată transformarea echivalentă a sistemului de coordonate Oxyz în sistemul de coordonate O'x'y'z' folosind matricea de transformare inversă corespunzătoare, $\mathbf{R_X}(-\pi/2)$: axa y se transformă în axa z', iar axa z se transformă în axa –y'. Punctul P, de coordonate 0,1,1 în sistemul Oxyz, se transformă în punctul P' de coordonate 0,-1,1 în sistemul O'x'y'z'.

Se pot verifica uşor rotaţiile cu un unghi de $\pi/2$ în raport cu celelalte axe de coordonate. Rotaţia cu $\pi/2$ faţă de axa y transformă sistemul de coordonate într-un sistem în care axa z se schimbă în axa x, iar axa x se schimbă în axa –z. Rotaţia cu $\pi/2$ faţă de axa z transformă sistemul de coordonate într-un sistem în care axa x se schimbă în axa y, iar axa y se transformă în axa –x.

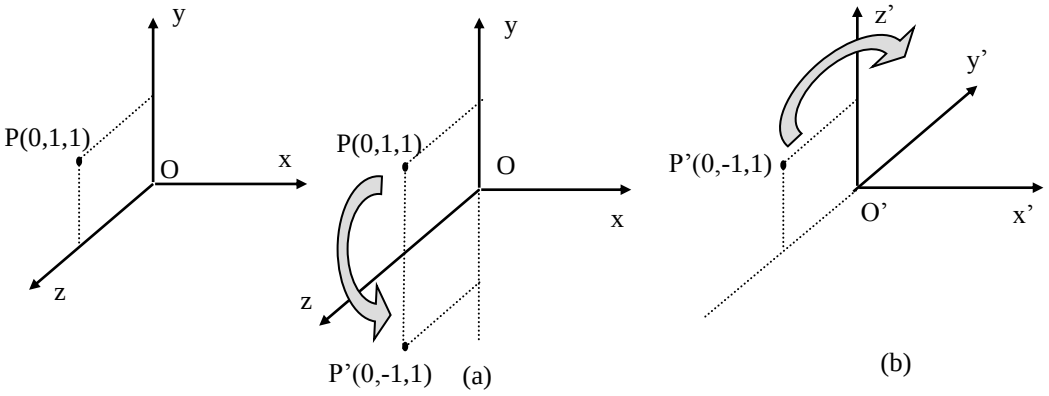


Fig. 3.6 Rotaţie cu $\pi/2$ în raport cu axa x.
(a) Rotaţia punctelor cu $\pi/2$. (b) Rotaţia sistemului de coordonate cu $-\pi/2$.

Pentru deducerea matricelor unor transformări complexe se procedează prin reducerea acestora la cazuri mai simple, pentru care se cunosc expresiile de calcul, folosind transformări primitive succesive. Iniţial se aplică o transformare ajutătoare pentru aducerea obiectelor într-o poziţie sau sistem de referinţă adecvat, se execută transformarea dorită, după care se revine la poziţia sau sistemul de referinţă iniţial prin transformarea inversă celei aplicate iniţial.

Câteva exemple prezentate în continuare vor preciza acest mod de realizare a transformărilor geometrice compuse.

3.1.5.1 Rotaţia faţă de o axă paralelă cu una din axele sistemului de referinţă

Un exemplu ilustrativ de compunere a transformărilor geometrice este calculul rotaţiei în raport cu o axă paralelă cu una din axele sistemului de referinţă.

Se consideră o dreaptă d paralelă cu axa z a sistemului, care intersectează planul Oxy în punctul $I(t_x,t_y,0)$. Transformarea de rotaţie a obiectelor faţă de această axă nu poate fi realizată folosind matricea dedusă în paragraful precedent (relaţia 3.10), care presupune rotaţia faţă de o axă a sistemului de coordonate.

De aceea, se aplică mai întâi tuturor punctelor o transformare ajutătoare, translaţia cu $\mathbf{T}(-t_x,-t_y,0)$.Acestă transformare poate fi intrepretată în două moduri: ca o schimbare a sistemului de referinţă Oxyz în sistemul O'x'y'z' , în care punctul I este transformat în punctul I'(0,0,0), sau ca o modificare a poziţiei tuturor punctelor din spaţiu, prin care punctul I este adus în originea sistemului de referinţă Oxyz.

Dacă se consideră prima interpretare, operaţiile se continuă în noul sistem de referinţă O'x'y'z', în care dreapta d se suprapune peste axa z', deci se poate obţine rotaţia dorită cu un unghi ρ folosind matricea de rotaţie 3.10. După aceasta, se revine la sistemul de referinţă iniţial Oxyz, printr-o transformare inversă, $\mathbf{T}(t_x, t_y, 0)$.

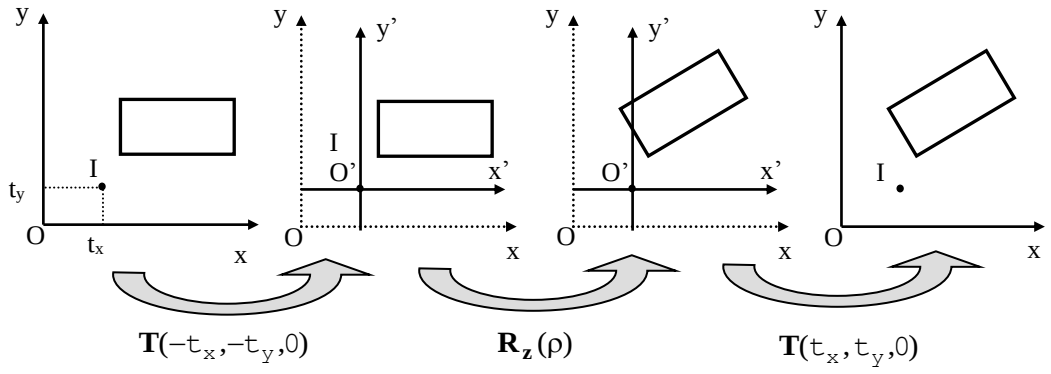
Rezultă matricea compusă de rotaţie cu unghiul ρ faţă de o dreaptă paralelă cu axa z care interectează planul Oxy în punctul $I(t_x, t_y,0)$:

$$\mathbf{R_d} = \mathbf{T}(t_x, t_y, 0)\mathbf{R_Z}(\rho)\mathbf{T}(-t_x, -t_y, 0)$$

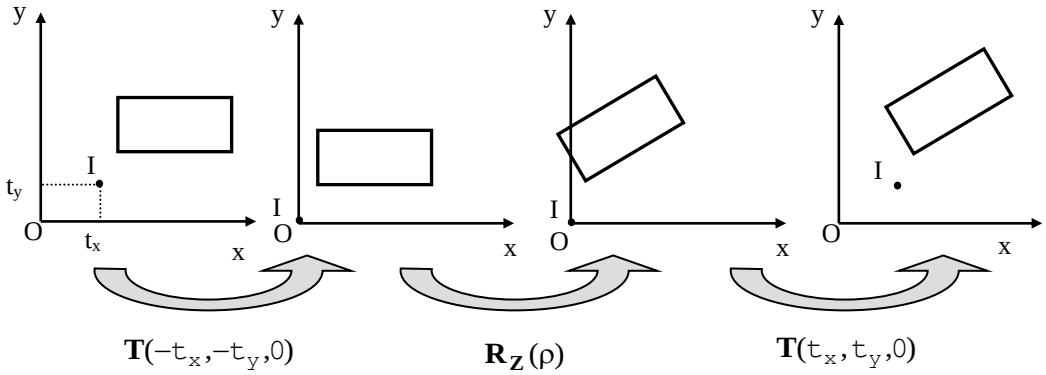
$$\mathbf{R_d} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\rho & -\sin\rho & 0 & 0 \\ \sin\rho & \cos\rho & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R_d} = \begin{bmatrix} \cos\rho & -\sin\rho & 0 & t_x(1-\cos\rho) + t_y \sin\rho \\ \sin\rho & \cos\rho & 0 & t_y(1-\cos\rho) - t_x \sin\rho \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.27)$$

Acest proces este descris în Fig. 3.7 (a) pentru un unghi de 30 grade, cu reprezentarea unei proiecţii în planul Oxy.



(a)



(b)

Fig. 3.7 Rotaţia faţă de o dreaptă paralelă ca axa z a sistemului de referinţă:
(a) considerând transformarea sistemelor de referinţă;
(b) considerând transformarea punctelor.

În Fig. 3.7 (b) se prezintă aceleaşi transformări succesive necesare pentru realizarea rotaţiei în raport cu o dreaptă paralelă cu axa z a sistemului, considerând că transformarea iniţială $\mathbf{T}(-t_x,-t_y,0)$ transformă toate punctele din sistemul de referinţă Oxyz, astfel încât punctul I se suprapune peste originea O. Se remarcă faptul că, în această situaţie, este doar o diferenţă de interpretare, operaţiile şi rezultatul acestora fiind identic.

3.1.5.2 Scalarea față de un punct oarecare în spațiu

Matricea de scalare din relaţia 3.4 efectuează scalarea faţă de originea sistemului de coordonate, prin care componentele x, y, z ale vectorului de poziţie OP al punctului P sunt multiplicate fiecare cu factorul de scalare corespunzător. Se poate defini o scalare faţă de un punct fix $F(x_f, y_f, z_f)$ din spaţiu, prin care componentele vectorului FP sunt multiplicate cu factorii de scalare corespunzători.

Matricea de scalare faţă de un punct oarecare se poate deduce printr-o metodă asemănătoare celei prezentate anterior, prin compunerea a trei transformări elementare deja definite.

1. Se execută o translaţie cu $\mathbf{T}(-x_f,-y_f,-z_f)$, prin care punctul fix F se suprapune peste originea sistemului de coordonate.
2. Se aplică o transformare de scalare cu matricea $\mathbf{S}(s_x, s_y, s_z)$.
3. Se efectuează o translaţie inversă celei aplicate la punctul 1, deci cu matricea $\mathbf{T}(x_f, y_f, z_f)$.

Rezultă matricea de scalare cu factorii de scară s_x, s_y, s_z faţă de un punct fix din spaţiu $F(x_f, y_f, z_f)$:

$$\mathbf{S_F} = \mathbf{T}(x_f, y_f, z_f)\mathbf{S}(s_x, s_y, s_z)\mathbf{T}(-x_f, -y_f, -z_f)$$

$$\mathbf{S_F} = \begin{bmatrix} 1 & 0 & 0 & x_f \\ 0 & 1 & 0 & y_f \\ 0 & 0 & 1 & z_f \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_f \\ 0 & 1 & 0 & -y_f \\ 0 & 0 & 1 & -z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{S_F} = \begin{bmatrix} s_x & 0 & 0 & x_f(1-s_x) \\ 0 & s_y & 0 & y_f(1-s_y) \\ 0 & 0 & s_z & z_f(1-s_z) \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.28)$$

3.1.5.3 Rotația față de o dreaptă oarecare în spațiu

Urmărind un raţionament asemănător cu cele anterioare, se poate deduce matricea de rotaţie cu un unghi δ fată de o dreaptă oarecare D din spaţiu, dată printr-un punct $D_0(x_0,y_0,z_0)$ şi vectorul unitate (versorul) **d**, având cosinuşii directori d_x,d_y,d_z ($\mathbf{d} = d_x\mathbf{i} + d_y\mathbf{j} + d_z\mathbf{k}$). Paşii de transformare sunt următorii:

1. Translaţia cu matricea $\mathbf{T}(-x_0,-y_0,-z_0)$, prin care punctul D_0 ajunge în originea sistemului de referinţă.
2. Alinierea dreptei D cu una din axele sistemului de referinţă, de exemplu, axa z . Alinierea cu una din axele de coordonate ale sistemului se efectuează prin două rotaţii în raport cu celelalte două axe.

Mai întâi se efectuează o rotaţe în raport cu axa x, cu un unghi α, astfel ca dreapta D să ajungă în planul Oxz. Pentru calculul unghiului α se consideră notaţiile din Fig. 3.8 (b).

OM este versorul dreptei D după translaţia efectuată în pasul precedent. M_x, M_y, M_z sunt intersecţiile planelor paralele cu planele sistemului de coordonate care trec prin punctul M, cu axele corespunzătoare (M_x este intersecţia planului paralel cu planul Oyz care trece prin M, cu axa x, etc.). M_{xy}, M_{yz}, M_{zx} sunt proiecţiile punctului M pe planele O_{xy}, O_{yz} şi, respectiv, O_{zx}. Se pot scrie următoarele relaţii:

$$OM_x = d_x;\ OM_y = d_y;\ OM_z = d_z$$

$$MM_x = d_{yz} = \sqrt{d_y^2 + d_z^2}$$

$$\cos\alpha = d_z / d_{yz} = d_z \Big/ \sqrt{d_y^2 + d_z^2};\ \sin\alpha = d_y / d_{yz} = d_y \Big/ \sqrt{d_y^2 + d_z^2} \qquad (3.29)$$

Rotaţia în raport cu axa x se efectuează în planul $MM_{xy}M_{zx}$. Punctul M, rotit cu unghiul α în planul $MM_{xy}M_{zx}$, se transformă în punctul M' aflat în planul O_{zx}, pe dreapta M_xM_{zx} paralelă cu axa Oz, astfel că $MM_x = M'M_x = d_{yz}$.

A doua etapă pentru alinierea versorului OM al dreptei D cu axa z este o rotaţie cu unghiul β, efectuată în planul Ozx, în raport cu axa y. Pentru calculul unghiului β se urmăresc notaţiile din Fig. 3.8 (c). În triunghiul dreptunghic $OM'M_x$

(unghiul drept în M_x) $OM_x = d_x$, $OM' = 1$, $M_xM' = d_{yz}$. Rezultă unghiul $OM'M_x$ (notat unghiul β):

$$\cos\beta = d_{yz} = \sqrt{d_y^2 + d_z^2}; \sin\beta = d_x \qquad (3.30)$$

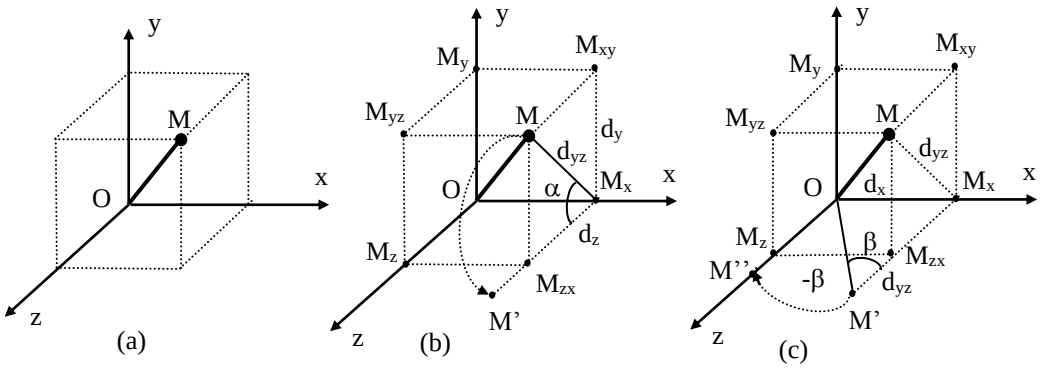


Fig. 3.8 Alinierea dreptei D cu axa z.

Rotaţia dreptei OM' în planul Ozx faţă de axa Oy transformă punctul M' în punctul M'' pe axa Oz printr-o rotaţie cu unghiul de valoare β în sens invers sensului pozitiv de rotaţie după axa y (a se revedea Fig. 3.3) Această transformare de aliniere a dreptei D cu axa z se exprimă matriceal astfel:

$$\mathbf{A_Z} = \mathbf{R_Y}(-\beta)\mathbf{R_X}(\alpha)$$

3. În acest pas se execută rotaţia dorită, cu unghiul δ în raport cu dreapta D, care este aliniată cu axa z a sistemului, deci se poate scrie:

$$\mathbf{R_Z}(\delta) = \begin{bmatrix} 1 & \cos\delta & -\sin\delta & 0 \\ 0 & \sin\delta & \cos\delta & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4. În pasul al patrulea se execută transformarea inversă celei de la pasul 2, deci cu o matrice $\mathbf{A_Z^{-1}} = \mathbf{R_X}(-\alpha)\mathbf{R_Y}(\beta)$.

5. În ultimul pas se execută transformarea de translaţie inversă, cu matricea $\mathbf{T}(x_0, y_0, z_0)$.

Rezultă matricea compusă de rotaţie faţă de dreapta D cu unghiul δ:

$$\mathbf{R_D} = \mathbf{T}(x_0, y_0, z_0)\mathbf{R_X}(-\alpha)\mathbf{R_Y}(\beta)\mathbf{R_Z}(\delta)\mathbf{R_Y}(-\beta)\mathbf{R_X}(\alpha)\mathbf{T}(-x_0, -y_0, -z_0) \qquad (3.31)$$

Valorile unghiurilor α şi β rezultă din cosinuşii directori ai axei de rotaţie D şi au valorile date de relaţiile 3.29 şi 3.30.

3.1.5.4 Transformarea de modelare

O scenă virtuală este compusă dintr-un număr oarecare de obiecte tridimensionale amplasate în diferite poziţii în scenă. Tehnica de modelare a scenelor de dimensiuni mari (ca număr de obiecte sau ca volum ocupat de scenă) se bazează pe amplasarea modelelor obiectelor tridimensionale în sistemul de referinţă universal al scenei virtuale.

Fiecare obiect este modelat într-un sistem de referinţă local, prin alegerea celei mai convenabile origini şi orientări a axelor de coordonate ale acestuia. De exemplu, un cub se modelează cel mai convenabil într-un sistem de referinţă ales cu centrul în centrul cubului şi cu axele orientate paralel cu laturile cubului. Dacă se presupune că latura cubului este egală cu 2 unităţi, atunci coordonatele vârfurilor cubului sunt (-1,-1,-1), (1,-1,-1), (1,1,-1), (-1,1,-1), (-1,-1,1), (1,-1,1), (1,1,1), (-1,1,1).

Modelul unui obiect, definit în sistemul de referinţă local, poate fi instanţiat (introdus) în scena virtuală printr-o transformare numită transformare de modelare (sau transformare de instanţiere). Mai mult, reprezentarea ierarhică a scenelor virtuale prin grafuri aciclice direcţionate, permite instanţierea de un număr oarecare de ori a fiecărui model, folosind matrice de transformare corespunzătoare. Acesta este modul cel mai frecvent de modelare a scenelor virtuale, care are suport în formatele şi limbajele de descriere a scenelor virtuale, cum este limbajul VRML.

În modelarea ierarhică a scenelor virtuale, operaţia de bază este transformarea de modelare, prin care toate punctele unui obiect (model tridimensional) sunt transformate din sistemul de referinţă local într-un alt sistem de referinţă, care poate fi sistemul de referinţă universal sau un alt sistem intermediar. Pentru început se consideră cea mai simplă transformare de modelare, dintr-un sistem de referinţă local în sistemul de referinţă universal.

În mod obişnuit, prima transformare care se aplică modelului este transformarea de scalare cu o matrice de scalare $\mathbf{S}(s_x,s_y,s_z)$, executată în sistemul de referinţă local (faţă de originea acestuia), prin care obiectul este adus la dimensiunile dorite ale instanţei respective.

Poziţia unei instanţe a obiectului în scenă se defineşte printr-o matrice de transformare care conţine submatricea de rotaţie **R**, care defineşte orientarea sistemului de referinţă local O'x'y'z' faţă de sistemul de referinţă universal Oxyz (al scenei virtuale) şi submatricea de translaţie **T**, care defineşte poziţia originii $O'(x_0,y_0,z_0)$ a sistemului de referinţă local în sistemul universal. Submatricea de rotaţie **R** poate fi specificată fie prin cosinuşii directori (c_{11}, c_{12}, c_{13}), (c_{21}, c_{22}, c_{23}), (c_{31}, c_{32}, c_{33}) ai axelor sistemului local faţă de sistemul de referinţă universal, fie printr-o succesiune de rotaţii faţă de axele de coordonate ale sistemului de referinţă local:

$$\mathbf{R} = \begin{bmatrix} c_{11} & c_{21} & c_{31} & 0 \\ c_{12} & r_{22} & c_{32} & 0 \\ c_{13} & c_{23} & c_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ sau } \quad \mathbf{R} = \mathbf{R_Y}(\chi)\mathbf{R_X}(\theta)\mathbf{R_Z}(\rho)$$

Rezultă matricea de transformare de modelare (instanţiere):

$$\mathbf{M_I} = \mathbf{T}(x_0, y_0, z_0)\mathbf{R}\,\mathbf{S}(s_x, s_{y,} s_z) \qquad (3.32)$$

Un punct P al obiectului în sistemul de referinţă model este transformat în punctul P_I în sistemul de referinţă universal prin instanţierea:

$$\mathbf{P_I} = \mathbf{M_I}\mathbf{P} \qquad (3.33)$$

Exemplul următor ilustrează crearea unei scene virtuale simple prin transformări de instanţiere.

ν **Exemplul 3.3**

Se consideră un cub ca latura de dimensiune 2, modelat într-un sistem de referinţă local ales cu centrul în centrul cubului şi cu axele orientate paralel cu laturile cubului şi se construieşte o scenă virtuală (în sistemul de referinţă universal) în care se instanţiază patru obiecte pornind de la modelul cub.

Prima instanţiere este chiar cubul din sistemul de referinţă model, deci sistemul de referinţă model are acelaşi centru şi aceeaşi orientare a axelor ca şi sistemul de referinţă universal; deci $C_1(0,0,0)$.

A doua instanţiere este un cub cu latura de 4 unităţi, cu orientarea sistemului de referinţă local cu axele de coordonate paralele cu axele sistemului de referinţă al scenei, amplasat cu centrul în punctul $C_2(0,-8,0)$.

A treia instanţiere este un paralelipiped dreptunghic, cu laturile de dimensiuni 4,2,4, amplasat cu centrul în poziţia $C_3(8,0,0)$. Sistemul de referinţă local are axa z' paralelă cu axa z a sistemului universal, iar axele x' şi y' sunt rotite cu un unghi de $\pi/4$ faţă de axa x şi, respectiv, y a sistemului universal.

A patra instanţiere este un paralelipiped dreptunghic, cu laturile de dimensiuni 4,2,4, amplasat cu centrul în poziţia $C_4(-8,0,0)$. Sistemul de referinţă local are axa z' paralelă cu axa z a sistemului universal, iar axele x' şi y' sunt rotite cu un unghi de $-\pi/4$ faţă de axa x şi, respectiv, y a sistemului universal.

Pentru crearea scenei se calculează pe rând cele trei matrice de instanţiere.

Matricea de instanţiere $\mathbf{M_1}$ este chiar matricea identitate: $\mathbf{M_1} = \mathbf{I}$.

Pentru calculul matricei de instanţiere $\mathbf{M_2}$, se deduc în mod foarte simplu matricele $\mathbf{S_2}=\mathbf{S}(2,2,2)$, $\mathbf{R_2}=\mathbf{I}$, şi $\mathbf{T_2}=\mathbf{T}(0,-8,0)$, deci: $\mathbf{M_2} = \mathbf{T}(0,-8,0)\,\mathbf{S}(2,2,2)$.

Pentru calculul matricei de instanţiere $\mathbf{M_3}$, se poate scrie $\mathbf{S_3}=\mathbf{S}(2,1,2)$, $\mathbf{R_3}=\mathbf{R_Z}(\pi/4)$, $\mathbf{T_3}=\mathbf{T}(8,0,0)$, deci: $\mathbf{M_3} = \mathbf{T}(8,0,0)\,\mathbf{R_Z}(\pi/4)\,\mathbf{S}(2,1,2)$.

Pemtru calculul matricei de instanţiere $\mathbf{M_4}$, se poate scrie: $\mathbf{S_4}=\mathbf{S}(2,1,2)$, $\mathbf{R_4}=\mathbf{R_Z}(-\pi/4)$, $\mathbf{T_4}=\mathbf{T}(-8,0,0)$, $\mathbf{M_4} = \mathbf{T}(-8,0,0)\,\mathbf{R_Z}(-\pi/4)\,\mathbf{S}(2,1,2)$.

Scena virtuală virtuală rezultată este redată în Fig. 3.9. Axele de coordonate reprezentate în imagine sunt axele sistemului de referinţă universal. În centrul sistemului de referinţă universal este reprezentat primul cub, cu aceleaşi dimensiuni ca cele ale modelului. Cubul din partea de jos a imaginii se obţine prin transformarea de instanţiere $\mathbf{P_2} = \mathbf{M_2}\,\mathbf{P}$; paralelipipedul din dreapta se obţine prin transformarea de instanţiere $\mathbf{P_3} = \mathbf{M_3}\,\mathbf{P}$; paralelipipedul din stânga se obţine prin transformarea de instanţiere $\mathbf{P_4} = \mathbf{M_4}\,\mathbf{P}$.

Reprezentarea de mai jos conţine şi o transformare de proiecţie perspectivă, pentru percepţia adâncimii care va fi explicată ulterior. Una din feţele cubului (cea mai apropiată de observator) este desenată ca suprafaţă de o culoare gri; celelalte feţe ale cubului sunt reprezentate numai prin muchiile lor (reprezentare numită "cadru de sârmă", *wireframe*)

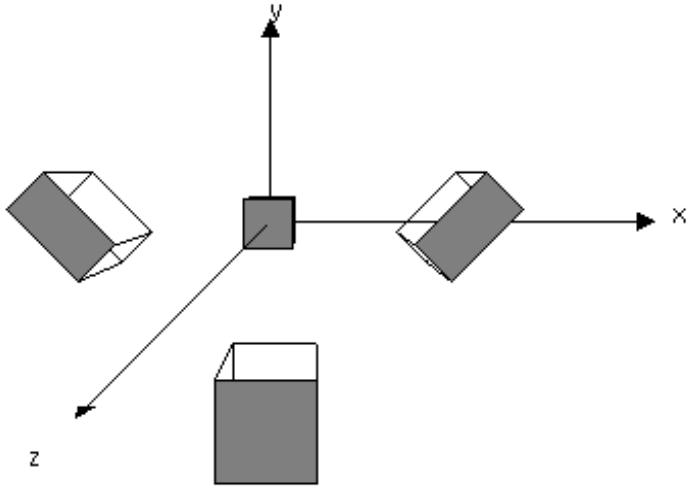


Fig. 3.9 Scenă virtuală rezultată prin instanţierea mai multor obiecte.

3.1.6 Alte transformări geometrice în spațiu

Printre alte transformări în spaţiul tridimensional care se pot aplica obiectelor, cele mai importante sunt transformarea de simetrie şi transformarea de forfecare.

Transformarea de simetrie. O transformare de simetrie (oglindire) a unui punct P faţă de un plan π calculează un punct P' aflat pe o dreaptă Δ perpendiculară pe planul π, la o distanţă egală cu distanţa lui P faţă de planul dat. Pentru cazul particular în care planul de simetrie este unul din planele sistemului de coordonate, simetricul unui punct este punctul care are coordonatele pe cele două axe ale planului egale şi coordonata pe cealaltă axă a sistemului de coordonate cu valoare negată.

Se pot deduce simplu matricele de transformare de simetrie relative la planele sistemului de referinţă:

$$\mathbf{S_{XY}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{S_{YZ}} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{S_{ZX}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.34)$$

Transformarea de simetrie faţă de un plan oarecare se poate calcula folosind o secvenţă de transformări elementare (E3.2).

Dacă se consideră transformarea de simetrie ca o transformare aplicată unui sistem de coordonate dat, sistemul de coordonate rezultat are convenţia de orientare a axelor inversă faţă de convenţia iniţială. De exemplu, sistemul drept Oxyz se transformă prin simetria $\mathbf{S_{xy}}$ într-un sistem de coordonate orientat după regula mâinii stângi (Fig. 3.10).

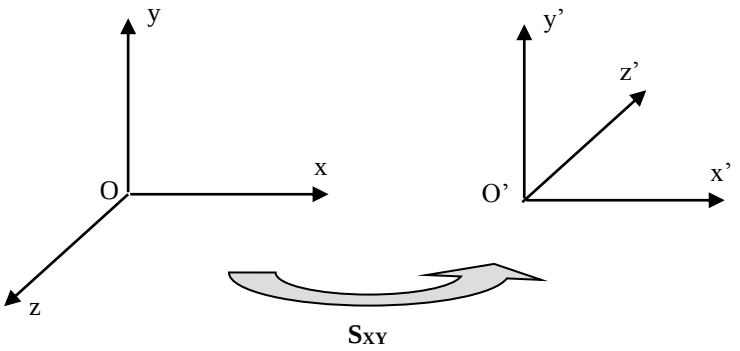


Fig. 3.10 Transformarea de simetrie $\mathbf{S_{XY}}$ transformă sistemul de coordonate drept Oxyz în sistemul de coordonate stâng O'x'y'z'.

Transformarea de forfecare (*shear*) este o transformare care modifică forma şi dimensiunea obiectului transformat. Transformările de forfecare cele mai simple sunt cele definite faţă axele sistemului de coordonate.

Forfecarea după axa x a sistemului de coordonate modifică un punct P(x,y,z) în punctul P'(x',y',z'), astfel că:

$$x' = x + ay + bz;\ y' = y;\ z' = z$$

În forma matriceală în coordonate omogene, forfecarea după axa x se scrie astfel:

$$\mathbf{P'} = \mathbf{P}\,\mathbf{F_X},\ \text{unde}\ \mathbf{F_X}(a,b) = \begin{bmatrix} 1 & a & b & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.35}$$

În mod similar se definesc matricele de forfecare pe axele y şi z:

$$\mathbf{F_Y}(c,d) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ c & 1 & d & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \qquad \mathbf{F_Z}(e,f) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ e & f & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(3.36)

Transformarea de forfecare este folosită în definirea sistemelor de proiecţie.

3.2 Transformări geometrice în plan

Uneori sunt necesare transformări geometrice în planul bidimensional. Exprimarea matriceală unitară a transaformărilor geometrice în plan se poate face într-un sistem de coordonate omogene definit în mod asemănător sistemului de coordonate omogene în spaţiu: unui punct P(x,y) în plan îi corespunde un punct P(X,Y,w), cu relaţiile între coordonate: $x = X / w; y = Y / w$, unde factorul de scară w este diferit de zero.

Transformările geometrice în planul bidimensional pot fi definite prin matrice de transformare de dimensiune 3x3. Pentru transformările geometrice primitive în plan, matricele de transformare se deduc simplu, ca o particularizare a transformărilor geometrice spaţiale. Acestea au următoarele expresii:

Translaţia cu valorile t_x şi t_y:

$$\mathbf{P'} = \mathbf{T}(t_x, t_y)\mathbf{P}, \text{ unde } \mathbf{T}(t_x, t_y) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.37}$$

Scalarea faţă de origine, cu factorii de scară s_x, s_y:

$$\mathbf{P'} = \mathbf{S}(s_x, s_y)\mathbf{P}, \text{ unde } \mathbf{S}(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.38}$$

Rotaţia faţă de origine cu un unghi θ:

$$\mathbf{P'} = \mathbf{R}(\theta)\mathbf{P}, \text{ unde } \mathbf{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.39}$$

Transformări complexe în plan se pot defini, la fel ca şi în spaţiu, prin compunerea (produs) de matrice de transformări primitive. Toate transformările între sisteme de referinţă în plan pot fi considerate cazuri particulare ale transformărilor corespunzătoare în spaţiu.

4

Sisteme de vizualizare

Un sistem de vizualizare este definit printr-o colecţie de sisteme de referinţă, convenţii de reprezentare şi relaţii (matriceale) de transformare care permit executarea unei succesiuni de transformări care au ca efect punerea în corespondenţă (mapping) a punctelor din sistemul de referinţă de modelare (*modelling coordinate space*) cu puncte ale suprafeţei de vizualizare (*view surface*).

Într-un sistem de vizualizare, generarea imaginii scenei virtuale este un proces complex care poate fi parcurs prin reprezentări în mai multe sisteme de referinţă, fiecare sistem de referinţă facilitând specificarea şi executarea anumitor operaţii (Fig. 4.1). În multe lucrări, generarea imaginii de mai este denumită şi vizualizarea sau redarea scenelor (*viewing*, *rendering*). Pe parcursul lucrării vor fi utilizaţi toţi aceşti termeni.

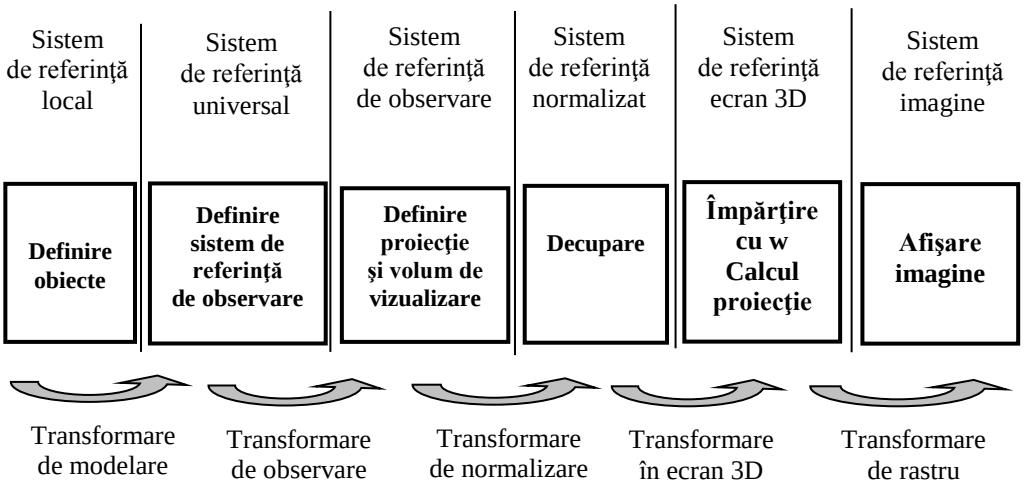


Fig. 4.1 Succesiunea operaţiilor de vizualizare.

Obiectele tridimensionale componente ale scenei sunt modelate şi reprezentate în sisteme de referinţă locale (sau de modelare), care au originea şi orientarea alese cel mai convenabil pentru reprezentarea obiectelor respective.

Pentru construirea scenei, obiectele specificate în sistemul de referinţă de modelare sunt instanţiate prin aplicarea unei succesiuni de transformări geometrice, care constitue transformarea de modelare (descrisă în paragraful 3.1.5.4). Prin transformarea de modelare se obţine descrierea obiectelor într-un sistem de referinţă unic al scenei virtuale, numit sistemul de referinţă universal. În sistemul de referinţă universal mai sunt definite şi alte elemente necesare redării realiste a scenelor, cum sunt sursele de lumină sau traiectoriile de modificare a poziţiei obiectelor mobile în scenă.

În *sistemul de referinţă universal* se defineşte un sistem de referinţă de observare (*view reference*), care specifică punctul şi direcţia din care este privită (observată) scena. Este evident că, pentru aceeaşi scenă, imaginea care se redă pe display depinde de punctul şi direcţia din care este observată scena, deci de felul în care este definit sistemul de referinţă de observare (care mai este denumit şi sistem de referinţă ochi- *eye coordinate space*). Transformarea obiectelor din sistemul de referinţă universal în sistemul de referinţă de observare este numită transformare de observare (sau de vizualizare – *view transform*; s-a evitat, însă, această traducere, datorită suprautilizării termenului de vizualizare).

În *sistemul de referinţă de observare* se poate face o analogie simplă cu modul în care este privită o scenă reală prin ochiul uman sau printr-un aparat fotografic (cameră). Poziţia şi orientarea sistemului de referinţă de observare corespund poziţiei şi orientării ochiului sau a camerei într-o scenă reală. Dintr-un punct de observare dat, numai o parte a scenei poate fi văzută, restul aflându-se în spatele sau în părţile laterale ale direcţiei de observare. Partea vizibilă din scenă (reală sau virtuală) este definită ca volum de vizualizare (*viewing volume*), care, în general, este o piramidă (sau un trunchi de piramidă) cu centrul în punctul de observare şi cu direcţia dată de direcţia de observare. În redarea scenelor, deschiderea unghiulară a piramidei de vizualizare defineşte câmpul de vizualizare (*field of view*), care este o caracteristică importantă a sistemelor grafice.

Deşi există şi unele dispozitive de afişare tridimensională, majoritatea sistemelor grafice afişează imaginile pe o suprafaţă plană a unui dispozitiv de afişare (display). De aceea, în succesiunea operaţiilor de vizualizare a obiectelor sau a scenelor tridimensionale, este necesară introducerea unei proiecţii, care converteşte spaţiul tridimensional într-o reprezentare bidimensională. În general, proiecţiile transformă spaţiul R^n în spaţiul R^m, unde $m < n$.

Modul în care se realizează proiecţia obiectelor tridimensionale se defineşte în sistemul de referinţă de observare prin mai mulţi parametri, dintre care cei mai importanţi sunt: tipul proiecţiei şi planul de vizualizare (*view plane*). Intersecţia dintre planul de vizualizare şi piramida de vizualizare defineşte o regiune plană numită fereastră de vizualizare (*view plane window*) în care sunt reprezentate toate imaginile bidimensionale ale obiectelor (sau părţilor de obiecte) vizibile din scenă.

În grafica tridimensională există mai multe posibilităţi de definire a sistemelor de vizualizare, de la definiri simple, utile în mici programe grafice, până la definiri

complete, care permit crearea simultană a mai multor imagini ale scenei, în ferestre de vizualizare diferite. În majoritatea acestor sisteme de vizualizare, din motive de eficienţă şi precizie a calculelor (care sunt explicate mai jos), transformarea de proiecţie se execută în două etape: prima etapă este o transformare de normalizare, care este o schimbare a sistemului de referinţă de la sistemul de referinţă de observare la sistemul de referinţă normalizat. Volumul de vizualizare definit în sistemul de referinţă de observare este transformat în sistemul de referinţă normalizat într-un volum canonic (în general, un paralelipiped dreptunghic).

În sistemul de referinţă normalizat (*normalized coordinates*) se execută decuparea obiectelor (*clipping*), astfel încât obiectele sau părţile din obiecte care se află în afara volumului de vizualizare (volumul canonic) sunt eliminate (ignorate pentru operaţiile următoare de redare, nu şterse din scenă).

Următoarea transformare geometrică este transformarea din sistemul normalizat în sistemul de referinţă ecran tridimensional (*three-dimensional sceen*), prin care fereastra definită în planul de vizualizare este transformată într-o regiune corespunzătoare zonei de afişare pe display, numită poartă (*viewport*), iar coordonata z se păstrează nemodificată. În sistemul de referinţă ecran 3D se calculează coordonatele tridimensionale ale punctelor din coordonatele omogene prin împărţirea cu w, după care se execută transformarea de rastru (*rasterization*). Transformarea de rastru calculează mulţimea pixelilor care aparţin unui segment sau poligon dat prin coordonatele vârfurilor. Prin această transformare (care se mai numeşte şi conversie de baleiere – *scan conversion*) spaţiul bidimensional continuu al porţii de afişare este convertit în spaţiul bidimensional discret al imaginii. În general, concomitent cu transformarea de rastru sunt executate şi alte operaţii: eliminarea suprafeţelor ascunse (*hidden surface removal*), umbrirea (*shading*), texturarea (*texturing*).

Aceasta este succesiunea cea mai generală a operaţiilor de vizualizare a scenelor, care este descrisă şi utilizată în momentul de faţă în grafică şi realitatea virtuală. Trebuie însă remarcat faptul că evoluţia tehnologică, deosebit de spectaculoasă în acest domeniu, a modificat, uneori substanţial, modul de abordare a generării imaginii obiectelor tridimensionale. De exemplu, posibilitatea de implementare eficientă (uneori chiar în hardware) a operaţiei de comparare a adâncimii punctelor a impus utilizarea algoritmului cunoscut sub numele de algoritm Z-buffer şi a condus la ignorarea completă a altor algoritmi de eliminare a suprafeţelor ascunse, care în sistemele mai vechi aveau o mare importanţă şi implicaţie asupra întregii succesiuni de operaţii grafice. Se pot da numeroase exemple de modul în care schimbările tehnologice au influienţat abordarea graficii tridimensionale, şi, probabil, astfel de influienţe vor continua să apară şi în viitor. De aceea, lucrarea îşi propune să prezinte aspectele esenţiale ale graficii utilizate în realitate virtuală, mai puţin influienţate de contextul tehnologic de moment.

Transformarea de observare

Transformarea de observare este analogă cu poziţionarea unei camere fotografice sau de înregistrare, prin care se observă o anumită zonă din scena virtuală. În sistemul de referinţă universal sunt reprezentate toate obiectele (modelele) scenei

virtuale, iar în fiecare cadru al imaginii (*frame*), este generată imaginea acelor obiecte din scenă care sunt vizibile din punctul de observare curent (*view point*).

Transformarea de observare este, aşadar, o schimbare a sistemului de coordonate de la sistemul de referinţă universal la sistemul de referinţă de observare (*view coordinate*). În majoritatea aplicaţiilor de realitate virtuală (simulatoare, aplicaţii în arhitectură, etc) scena virtuală reprezentată în sistemul de referinţă universal este menţinută într-o poziţie constantă (cel mult existând anumite obiecte mobile în scenă), iar observatorul este cel care îşi modifică poziţia şi orientarea, şi în funcţie de acestea se redă imaginea corespunzătoare pe ecran. Direcţia de observare a scenei este, în majoritatea sistemelor de vizualizare, direcţia axei z a sistemului de referinţă de observare.

Localizarea şi orientarea sistemului de referinţă de observare $O_V x_V y_V z_V$ în raport cu sistemul de referinţă universal Oxyz poate fi descrisă printr-o matrice de transformare $\mathbf{M}$ de forma generală:

$$\mathbf{M} = \mathbf{T_V R_V} = \begin{bmatrix} 1 & 0 & 0 & x_V \\ 0 & 1 & 0 & y_V \\ 0 & 0 & 1 & z_V \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_{11} & c_{21} & c_{31} & 0 \\ c_{12} & c_{22} & c_{32} & 0 \\ c_{13} & c_{23} & c_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c_{11} & c_{21} & c_{31} & x_V \\ c_{12} & c_{22} & c_{32} & y_V \\ c_{13} & c_{23} & c_{33} & z_V \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (4.1)$$

Matricea componentă $\mathbf{T_V}$ este matricea de translaţie a sistemului de referinţă de observare în raport cu sistemul de referinţă universal: x_V, y_V, z_V sunt coordonatele punctului de observare O_V în sistemul Oxzy. Matricea de rotaţie $\mathbf{R_V}$ reprezintă orientarea sistemului de referinţă de observare faţă de sistemul de referinţă universal: coeficienţii c_{11}, c_{12}, c_{13} sunt cosinuşii directori ai axei $O_V x_V$ faţă de axele Ox, Oy, respectiv Oz; coeficienţii c_{21}, c_{22}, c_{23} sunt cosinuşii directori ai axei $O_V y_V$ faţă de axele Ox, Oy, respectiv Oz; coeficienţii c_{31}, c_{32}, c_{33} sunt cosinuşii directori ai axei $O_V z_V$ faţă de axele Ox, Oy, respectiv Oz.

Fiind dată matricea de poziţionare şi orientare a sistemului de referinţă observator relativ la sistemul de referinţă universal $\mathbf{M}$, un punct oarecare P(x,z,y) din sistemul Oxyz se transformă în punctul $P_V(x_V,y_V,z_V)$ în sistemul de referinţă observator prin înmulţire cu matricea inversă, deci:

$$\mathbf{P_V} = \mathbf{M^{-1}P} = \mathbf{R_V^{-1}T_V^{-1}P} = \mathbf{M_V P} \qquad (4.2)$$

unde matricea de transformare de observare $\mathbf{M_V}$ este inversa matricii de localizare a sistemului de referinţă de observare relativ la sistemul de referinţă universal: $\mathbf{M_V} = \mathbf{M^{-1}}$.

Matricea de rotaţie mai poate fi precizată şi printr-o succesiune de rotaţii în raport cu axele sistemului de referinţă universal. Cea mai obişnuită convenţie pentru ordinea de specificare a rotaţiilor este: ruliu cu unghiul ρ (după axa z), tangaj cu unghiul θ (după axa x) şi giraţie cu unghiul χ (faţă de axa y). În această situaţie:

$$\mathbf{R_V} = \mathbf{R_Y}(\chi)\mathbf{R_X}(\theta)\mathbf{R_Z}(\rho)$$

şi rezultă:

$$\mathbf{P_V} = \mathbf{R_V^{-1}T_V^{-1}P} = \mathbf{R_Z}(-\rho)\mathbf{R_X}(-\theta)\mathbf{R_Y}(-\chi)\mathbf{T}(-x_V,-y_v,-z_v)\mathbf{P} \qquad (4.3)$$

În aceste relaţii de transformare de observare s-a presupus că sistemul de referinţă de observare este tot un sistem drept, la fel ca şi sistemul de referinţă universal. Există însă şi sisteme de vizualizare în care se defineşte sistemul de referinţă de observare ca un sistem orientat după regula mâinii stângi. Pentru schimbarea sistemului de referinţă de observare drept în sistem de referinţă stâng, se adaugă o transformare de oglindire (simetrie), de exemplu cu matricea $\mathbf{S_{XY}}$, iar matricea de transformare de observare devine $\mathbf{M'_V}$, unde:

$$\mathbf{M'_V = S_{XY} M_V}$$

Dat fiind că nu există o predilecţie certă pentru una din aceste convenţii posibile (sistem de referinţă de observare drept sau stâng), este necesar să fie precizată de fiecare dată convenţia folosită, ceea ce se va face şi pe parcursul acestei lucrări.

În deducerea relaţiilor 4.2 şi 4.3 s-a presupus că este cunoscută matricea **M** de localizare a sistemului de referinţă de observare faţă de sistemul de referinţă universal şi atunci matricea de transformare de observare $\mathbf{M_V}$ se calculează ca matrice inversă a acesteia. Există situaţii în care matricea **M** nu este cunoscută şi atunci matricea de transformare de observare $\mathbf{M_V}$ (sau $\mathbf{M'_V}$) trebuie să fie calculată prin diferite transformări geometrice şi raţionamente. În exemplul următor se urmăreşte un raţionament de calcul al transformării de observare, pentru o anumită cerinţă de observare a scenei.

ν Exemplul 4.1

Se poate defini un sistem de referinţă de observare pornind de la alte condiţii decât cele prezentate mai sus. De exemplu, se cere definirea matricii de transformare de observare în situaţia în care obiectul care trebuie să fie văzut se află în centrul sistemului de referinţă universal, iar punctul de observare este un punct dat în spaţiul tridimensional, $E(x_E, y_E, z_E)$.

Pentru ca obiectul aflat în centrul sistemului de referinţă universal să fie văzut din punctul E dat, se defineşte sistemul de referinţă de observare drept cu centrul în punctul E, şi cu axa z_V orientată pe dreapta OE (cu sensul de la O către E). Direcţia de observare într-un sistem de referinţă drept este definită (în majoritatea sistemelor de vizualizare) în sens invers direcţiei axei z, deci către centrul O.

Calculele de transformare se efectuează mai uşor dacă se dau coordonatele sferice ale punctului E: distanţa μ a punctului E faţă de origine şi unghiurile θ şi ϕ (Fig. 4.2). Coordonatele carteziene ale punctului E pot fi calculate în funcţe de coordonatele sferice:

$$x_E = \mu \sin\phi\cos\theta$$

$$y_E = \mu \sin\phi\cos\theta$$

$$z_E = \mu \cos\phi$$

Pentru transformarea unui obiect din sistemul de referinţă universal în sistemul de referinţă de observare astfel definit, sunt necesare trei transformări elementare. În Fig. 4.2 sunt reprezentate primele două transformări componente ale transformării de observare.

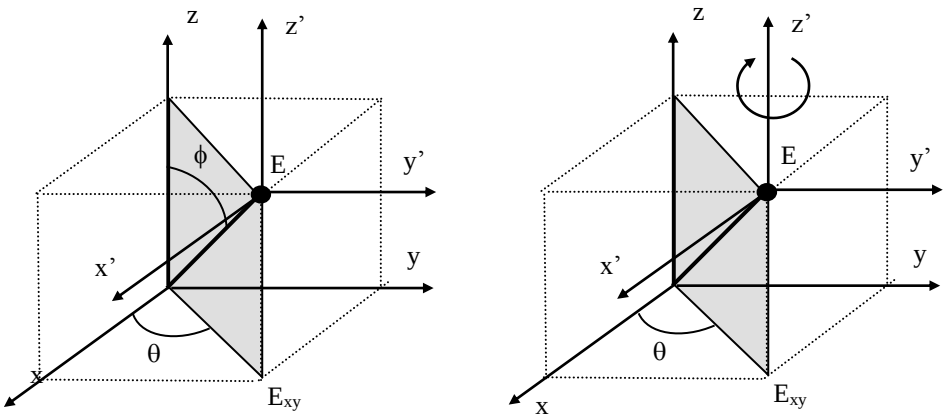


Fig. 4.2 Transformarea de observare: (a) translaţia (b) rotaţia în raport cu axa z'.

Translaţia $\mathbf{T_1} = \mathbf{T}(-x_E, -y_E, -z_E)$ transformă sistemul de referinţă universal într-un sistem de referinţă cu centrul în punctul E şi cu axele x', y', z' paralele cu axele sistemului de referinţă universal.

$$\mathbf{T_1} = \begin{bmatrix} 1 & 0 & 0 & -\mu\cos\theta\,\sin\phi \\ 0 & 1 & 0 & -\mu\sin\theta\,\sin\phi \\ 0 & 0 & 1 & -\mu\cos\phi \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Orientarea sistemului de referinţă de observare astfel încât axa z' să se suprapună peste dreapta OE se obţine prin două rotaţii.

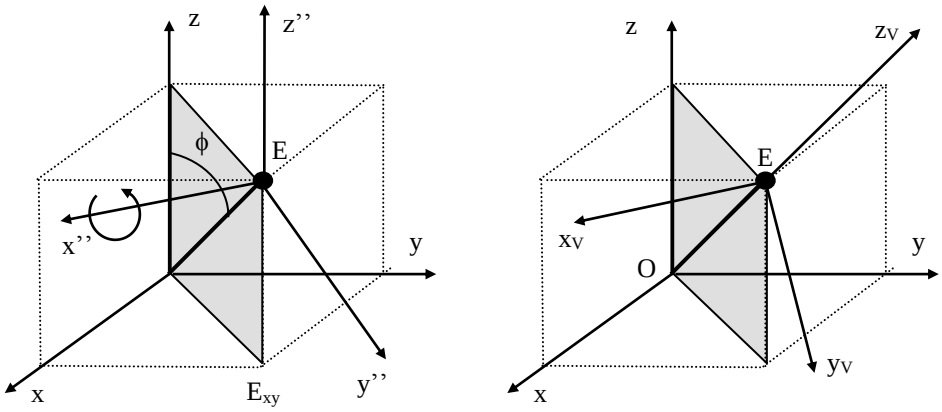
Prima rotaţie se efectuează în raport cu axa z' cu un unghi de 90° – θ în sens negativ, astfel încât noua axă x'' obţinută să fie normală la planul care trece prin axa z şi punctul E (planul OEE_{xy}) (Fig. 4.2(b)). Această operaţie este echivalentă cu o rotaţie în sens direct aplicată asupra punctelor, deci cu matricea $\mathbf{R_2} = \mathbf{R_Z}(90°-\theta)$, unde

$$\mathbf{R_2} = \begin{bmatrix} \sin\theta & -\cos\theta & 0 & 0 \\ \cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

După această rotaţie, sistemul observator are axa x'' perpendiculară pe planul planul OEE_{xy} şi axa Ez'' în planul OEE_{xy}.

A doua rotaţie este o rotaţie a sistemului de coordonate Ex''y''z'' în sens negativ cu un unghi egal cu ϕ în raport cu axa Ex'', pentru a suprapune axa Ez'' peste dreapta OE (Fig. 4.2(a)). Acestă rotaţie este echivalentă cu o rotaţie inversă aplicată obiectelor, deci cu matricea $\mathbf{R_3} = \mathbf{R_X}(\phi)$:

$$\mathbf{R_3} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



(c) (d)

Fig. 4.2 Transformarea de observare:
(c) rotaţia în raport cu axa x''(d) sistemul de referinţă rezultat

Rezultă matricea de transformare de observare:

$$\mathbf{M_V} = \mathbf{R_X}(\phi)\mathbf{R_Z}(90° - \theta)\mathbf{T}(-x_E, -y_E, -z_E)$$

$$\mathbf{M_V} = \begin{bmatrix} \sin\theta & -\cos\theta & 0 & 0 \\ \cos\theta\cos\phi & \sin\theta\cos\phi & -\sin\phi & 0 \\ \cos\theta\sin\phi & \sin\theta\sin\phi & \cos\phi & -\mu \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformarea unui punct P(x,y,z) din sistemul de referinţă universal în punctul $P(x_V,y_V,z_V)$ în sistemul de referinţă observator se obţine astfel:

$$\begin{bmatrix} x_V & y_V & z_V & 1 \end{bmatrix}^T = \mathbf{M_V} \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T$$

În Fig. 4.3 este reprezentată imaginea unui cub cu centrul în origine şi laturile egale cu 2 şi paralele cu axele de coordonate, observat din punctul E(10,10,10). Pentru punctul E:

$\mu = 10\sqrt{3}$

$\cos\theta = \sqrt{2}/2, \sin\theta = \sqrt{2}/2, \theta = 45^\circ$

$\cos\phi = \sqrt{3}/3, \sin\phi = \sqrt{6}/3, \phi = 54^\circ 45'$

Matricea de transformare de observare are valoarea:

$$\mathbf{M_V} = \begin{bmatrix} \sqrt{2}/2 & -\sqrt{2}/2 & 0 & 0 \\ \sqrt{6}/6 & \sqrt{6}/6 & -\sqrt{6}/3 & 0 \\ \sqrt{3}/3 & \sqrt{3}/3 & \sqrt{3}/3 & -10\sqrt{3} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

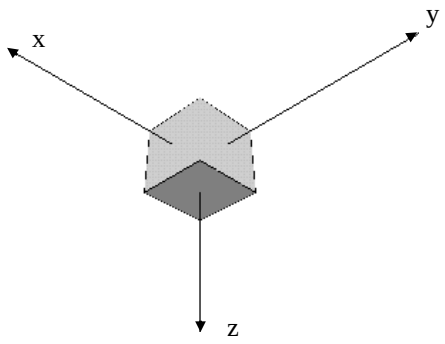


Fig. 4.3 Exemplu de observare către centrul sistemului de referinţă universal.

Centrul O al sistemului de referinţă universal se transformă în punctul $O'(0,0,-9\sqrt{3})$ în sistemul de referinţă de observare, iar vârful cubului P(1,1,1), se transformă în punctul $P'(0,0,-10\sqrt{3})$. Direcţia de observare trece printr-un vârf al cubului şi prin centrul lui. ν

Transformarea de proiecţie

Proiecţia unui obiect tridimensional pe un plan (numit plan de proiecţie sau plan de vizualizare – *view plane*) este formată din mulţimea punctelor de intersecţie între acest plan şi dreptele numite proiectori, care pornesc dintr-un punct fix numit centru de proiecţie şi trec prin fiecare punct al obiectului.

În funcţie de distanţa centrului de proiecţie faţă de planul de proiecţie, proiecţiile plane se clasifică în proiecţii perspectivă şi proiecţii paralele. În proiecţiile perspectivă centrul de proiecţie se află la distanţă finită faţă de planul de vizualizare, iar în proiecţiile paralele centrul de proiecţie este la infinit. În cazul proiecţiilor paralele, proiectorii sunt drepte paralele şi au o direcţie comună, numită direcţie de proiecţie.

În grafica tridimensională se foloseşte cel mai frecvent proiecţia perspectivă, deoarece această proiecţie redă cel mai bine efectul de descreştere a dimensiunilor obiectelor odată cu creşterea distanţei acestora faţă de observator.

Modul de definire a punctului de observare al scenei, al centrului de proiecţie şi al planului de vizualizare depind de sistemul de vizualizare folosit. Un studiu atent al sistemelor de vizualizare este necesar din mai multe motive. Chiar dacă un utilizator nu trebuie să proiecteze el însuşi sistemul de vizualizare, exploatarea oricărui sistem necesită o înţelegere precisă a modului în care diferiţi parametri determină imaginea obţinută pe display. Combinaţiile posibile sunt în număr atât de mare, încât şansa de a potrivi diferiţi parametri ai unui sistem de vizualizare, fără a fi cunoscute condiţiile în care lucrează acesta, este infimă.

Proiecția paralelă

În prioecţia paralelă, centrul de proiecţie este la infinit, iar toate dreptele de proiecţie sunt paralele între ele, având o direcţie numită direcţia de proiecţie.

Proiecţiile paralele sunt foarte rar folosite în grafica destinată sistemelor de realitate virtuală, dat fiind că imaginea rezultată nu dă informaţii asupra distanţei obiectelor în scenă.

Planul de proiecţie al unei proiecţii paralele poate fi perpendicular pe direcţia de proiecţie, şi atunci proiecţia se numeşte proiecţie ortografică, sau poate avea o inclinare oarecare faţă de direcţia de proiecţie, şi atunci proiecţia se numeşte proiecţie oblică.

Dintre proiecţiile ortografice, cele mai utilizate sunt proiecţiile ortografice în care planul de proiecţie este unul din planurile sistemului de coordonate. Proiecţiile ortografice sunt folosite frecvent în desenul tehnic şi, de asemenea, ca o etapă intermediară în definirea sistemelor de vizualizare normalizate, care vor fi prezentate în paragraful următor. Matricea de transformare de proiecţie ortografică în planul z = 0 are următoarea expresie:

$$\mathbf{P_Z} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

Un punct P(x,y,z) se transformă în punctul P'(x,y,0) prin proiecţie ortografică în planul z=0, care are aceleaşi coordonate x şi y, iar coordonata z are valoarea zero.

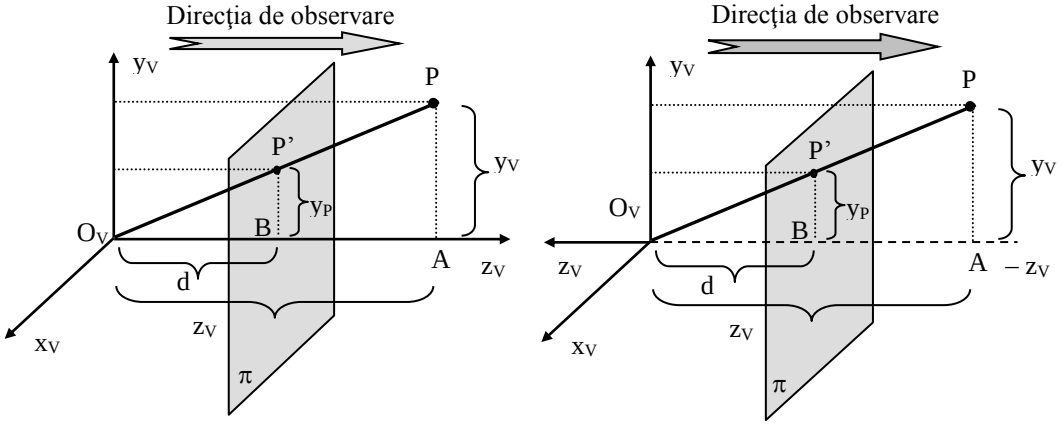
În mod similar, se definesc matricile de proiecţie ortografică în planele x = 0 şi y = 0:

$$\mathbf{P_X} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{P_Y} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

Proiecţia perspectivă

Există mai multe modalităţi de definire a unei proiecţii perspectivă, de la o definire minimală, prin care se poate realiza proiecţia punctelor din spaţiu pe un plan, până la definirea adoptată în sisteme de vizualizare complexe, în care se specifică volumul de vizualizare, centrul de proiecţie şi alţi parametri. În continuare, vor fi prezentate câteva sisteme de vizualizare cu proiecţie perspectivă, începând cu cel mai simplu sistem şi până la sistemul complet de vizualizare care se conformează standardelor GKS şi PHIGS.

Proiecţia perspectivă se poate defini mai intuitiv pornind de la sistemul de observare considerat ca un sistem de referinţă stâng. Centrul de proiecţie se fixează în centrul sistemului de referinţă de observare O_V, direcţia de observare (de privire) este în sensul pozitiv al axei O_Vz_V, iar planul de vizualizare π este un plan perpendicular pe axa O_Vz_V, la distanţa d faţă de centru (Fig. 4.4(a)).



(a) Sistem de referinţă de observare stâng (b) Sistem de referinţă de observare drept

Fig. 4.4 Proiecţia perspectivă definită prin centrul de proiecţie O_V şi planul de proiecţie π perpendicular pe axa O_Vz_V.

Punctul $P(x_V,y_V,z_V)$ în sistemul de referinţă de observare se proiectează în punctul $P'(x_P,y_P,z_P)$, aflat la intersecţia dreptei O_VP cu planul de vizualizare. Din triunghiurile asemenea $O_VP'A$ şi O_VPB, rezultă:

$$y_P = \frac{d\, y_V}{z_V}$$

În mod asemănător se poate deduce valoarea lui x_P:

$$x_P = \frac{d\, x_V}{z_V}$$

(4.6)

Aceste relaţii se pot exprima matriceal prin definirea matricei de proiecţie perspectivă $\mathbf{M_P}$ astfel:

$$\mathbf{M_P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \tag{4.7}$$

Transformarea de proiecţie se poate scrie atunci:

$$\mathbf{P' = M_P P}, \quad \begin{bmatrix} X_P \\ Y_P \\ Z_P \\ w_P \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{bmatrix} X_V \\ Y_V \\ Z_V \\ w_V \end{bmatrix}$$

Rezultă:

$$\begin{cases} X_P = X_V \\ Y_P = Y_V \\ Z_P = Z_V \\ w_P = Z_V / d \end{cases} \text{ şi } \begin{cases} x_P = X_P / w_P = d\, X_V / Z_V = d\, x_V / z_V \\ y_P = Y_P / w_P = d\, Y_V / Z_V = d\, y_V / z_V \\ z_P = Z_P / w_P = d \end{cases} \tag{4.8}$$

Se observă că expresiile lui x_P şi y_P şi z_P obţinute prin calcul matriceal sunt identice cu cele din relaţiile 4.6, rezultate din construcţia geometrică prezentată în Fig 4.4(a). Proiecţia perspectivă înseamnă deci o împărţire a coordonatelor x şi y cu distanţa z, şi, într-adevăr, în proiecţia perspectivă imaginea unui obiect este cu atât mai mică cu cît se află la distanţă mai mare de punctul de observare.

În acest sistem de vizualizare simplu s-a definit sistemul de referinţă de observare ca un sistem stâng. Se poate defini acest sistem ca un sistem de referinţă drept, iar direcţia de observare îndreptată spre $-z_V$. Planul de vizualizare este poziţionat perpendicular pe axa z, la distanţă egală cu d faţă de origine (Fig. 4.4(b)), d având valoare negativă. Urmând un raţionament asemănotor cu cel de mai sus, rezultă matricea de proiecţie perspectivă cu aceeaşi expresie 4.7, în care d are o valoare negativă. Coordonatele punctului transformat sunt date de relaţiile 4.8. Trebuie precizat că în această convenţie se proiectează pe planul de vizualizare punctele aflate în direcţia de observare, deci cu z_V negativ.

Se remarcă echivalenţa rezultatelor obţinute, iar convenţia de sistem de referinţă de observare stâng este utilă doar pentru a se urmări mai uşor raţionamentul de calcul.

Sistemul de referinţă normalizat

Sistemul de proiecţie simplu definit mai sus, în care toate obiectele din scenă sunt proiectate pe planul de vizualizare, are mai multe inconveniente şi nu poate fi folosit decât în aplicaţii foarte simple, în care se proiectează obiecte aflate tot timpul în câmpul de observare. Nu aceasta este situaţia imaginilor generate în realitatea virtuală,

în care se explorează o scenă virtuală de dimensiuni mari, din care numai o parte este vizibilă şi deci redată pe ecran.

Transformarea de proiecţie perspectivă este o transformare costisitoare, datorită faptului că necesită operaţii de împărţire, care consumă timp de calcul ridicat, dacă este executată prin program, sau circuite complexe, dacă este implementată hardware. Execuţia ei pentru toate obiectele scenei, inclusiv pentru cele care nu sunt vizibile, deci nu sunt reprezentate pe display şi nu necesită coordonatele în planul de vizualizare, este costisitoare şi inutilă. În plus, operaţia de decupare (*clipping*) a părţilor din obiectele tridimensionale care nu sunt vizibile dintr-un anumit punct de observare, nu se poate executa eficient după proiecţie şi este necesară execuţia ei înainte de proiectarea coordonatelor în planul de vizualizare. Asupra acestui aspect se va reveni cu ocazia descrierii operaţiei de decupare.

Este evident că, pentru redarea eficientă a scenelor tridimensionale, este necesară definirea unui volum de vizualizare (*view volume*), cunoscut sub numele de piramida de vizualizare sau trunchiul de piramidă de vizualizare (*frustum*). Trunchiul de piramidă de vizualizare se defineşte în sistemul de referinţă de observare. În forma cea mai simplă, trunchiul de piramidă de vizualizare este mărginit de patru plane laterale care trec prin punctul de observare O_V şi de două plane perpendiculare pe axa O_Vz_V, planul apropiat (*near*) pe care se execută proiecţia, şi planul depărtat (*far*). În figura Fig. 4.5 este prezentat volumul de vizualizare într-un sistem de referinţă de observare stâng şi direcţia de observare către z_V pozitiv.

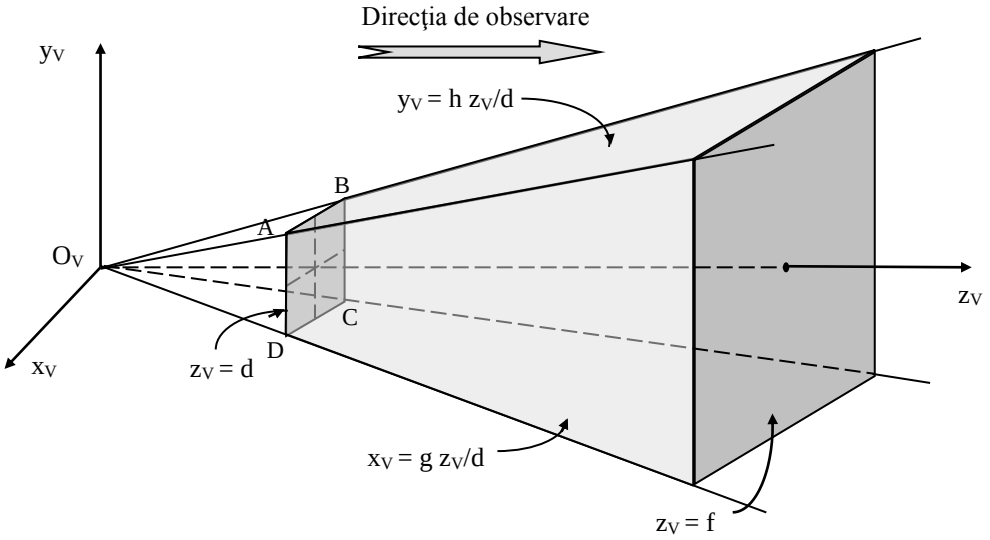


Fig. 4.5 Definirea trunchiului de piramidă de vizualizare (frustum) în sistemul de referinţă de observare stâng.

Intersecţia dintre cele patru plane laterale şi planul de vizualizare (care, în această definiţie, este şi planul apropiat de vizibilitate) determină o suprafaţă dreptunghiulară ABCD în planul de vizualizare în care vor fi proiectate toate obiectele vizibile ale scenei. Această suprafaţă este fereastra de vizualizare (*view plane window*). Ecuaţiile celor şase plane ale trunchiului de piramidă regulată sunt:

$$\begin{cases} x_V = \pm g\, z_V / d \\ y_V = \pm h\, z_V / d \\ z_V = d \\ z_V = f \end{cases}$$

(4.9)

unde: 2g – dimensiunea laturii orizontale a ferestrei de vizualizare (latura AB)
2h – dimensiunea laturii verticale a ferestrei de vizualizare (latura BC)
d – distanţa planului de vizualizare şi a planului apropiat
f – distanţa planului depărtat.

Semnificaţia volumului de vizualizare este evidentă: dintre toate obiectele din scena virtuală, sunt vizibile şi redate în fereastra de vizualizare numai acele obiecte sau părţi din obiecte care sunt cuprinse în acest volum. Trunchiul de piramidă de vizualizare defineşte în acest fel şi volumul de decupare (*clipping*), operaţie care se execută într-unul din sistemele de referinţă care se definesc în cadrul transformării de proiecţie, şi anume sistemul de referinţă normalizat. Acest sistem de referinţă se mai numeşte şi sistem de referinţă canonic. Motivele pentru care decuparea se efectuează cel mai eficient în sistemul de referinţă normalizat vor fi explicate ulterior.

Sistemul de referinţă normalizat este un sistem de coordonate tridimensional în care trunchiul de piramidă de vizualizare se transformă într-un volum canonic, care, în general, este un paralelipiped dreptunghic. Pentru sistemul de proiecţie definit în acest paragraf, volumul canonic este un paralelipiped dreptunghic cu latura bazei egală cu 2 şi înălţimea 1 (după axa z_N).

Introducerea acestui nou sistem de referinţă permite abordarea transformării de proiecţie perspectivă în două etape: o transformare din sistemul de referinţă de observare în sistemul de referinţă normalizat (numită transformare de normalizare), urmată de calculul proiecţiei prin împărţirea cu w a coordonatelor omogene. Între cele două etape ale transformării de proiecţie perspectivă se execută o operaţie importantă în grafica tridimensională, decuparea obiectelor relativ la volumul de vizualizare normalizat (volumul canonic).

Pentru transformarea de normalizare (transformarea de la sistemul de referinţă de observare la sistemul de referinţă normalizat), se impun următoarele cerinţe:

(1) Se normalizează coordonata z_N, care reprezintă adâncimea, astfel încât să se obţină precizie maximă în operaţiile de comparaţie a adâncimii punctelor.
(2) Liniile din sistemul de referinţă observator trebuie să se transforme în linii în sistemul de referinţă normalizat.
(3) Suprafeţele plane din sistemul de referinţă observator trebuie să se transforme în suprafeţe plane în sistemul de referinţă normalizat.

S-a demonstrat [New81] că aceste cerinţe sunt satisfăcute de transformarea:

$$z_N = a + b/z_V$$

unde a şi b sunt constante. Aceste constante se determină din următoarele constrângeri:

- Se alege b < 0, astfel ca z_N să crească atunci când creşte z_V. Această condiţie conservă noţiunea intuitivă de adâncime: un punct are coordonata z_N cu atât mai mare cu cât este mai departe de observator.
- Se normalizează domeniul de variaţie al lui z_N, astfel ca domeniul $z_V \in [d,f]$ să se transforme în domeniul $z_N \in [0,1]$.

Rezultă următoarele relaţii care definesc transformarea de normalizare:

$$\begin{cases} x_N = \dfrac{d\, x_V}{g\, z_V} \\ y_N = \dfrac{d\, y_V}{h\, z_V} \\ z_N = \dfrac{f}{f-d}\left(1 - \dfrac{d}{z_V}\right) \end{cases} \tag{4.10}$$

Coordonatele omogene ale unui punct în sistemul normalizat au expresiile:

$$\begin{cases} X_N = d x_V / g \\ Y_N = d y_V / h \\ Z_N = z_V f /(f-d) - df/(f-d) \\ w_N = z_V \end{cases} \tag{4.11}$$

Din aceste expresii se deduce matricea de transformare de normalizare $\mathbf{M_N}$ care efectuează transformarea de la sistemul de referinţă observator la sistemul de referinţă normalizat:

$$\mathbf{M_N} = \begin{bmatrix} d/g & 0 & 0 & 0 \\ 0 & d/h & 0 & 0 \\ 0 & 0 & f/(f-d) & -df/(f-d) \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \begin{bmatrix} X_N \\ Y_N \\ Z_N \\ w_N \end{bmatrix} = \mathbf{M_N} \begin{bmatrix} x_V \\ y_V \\ z_V \\ 1 \end{bmatrix} \tag{4.12}$$

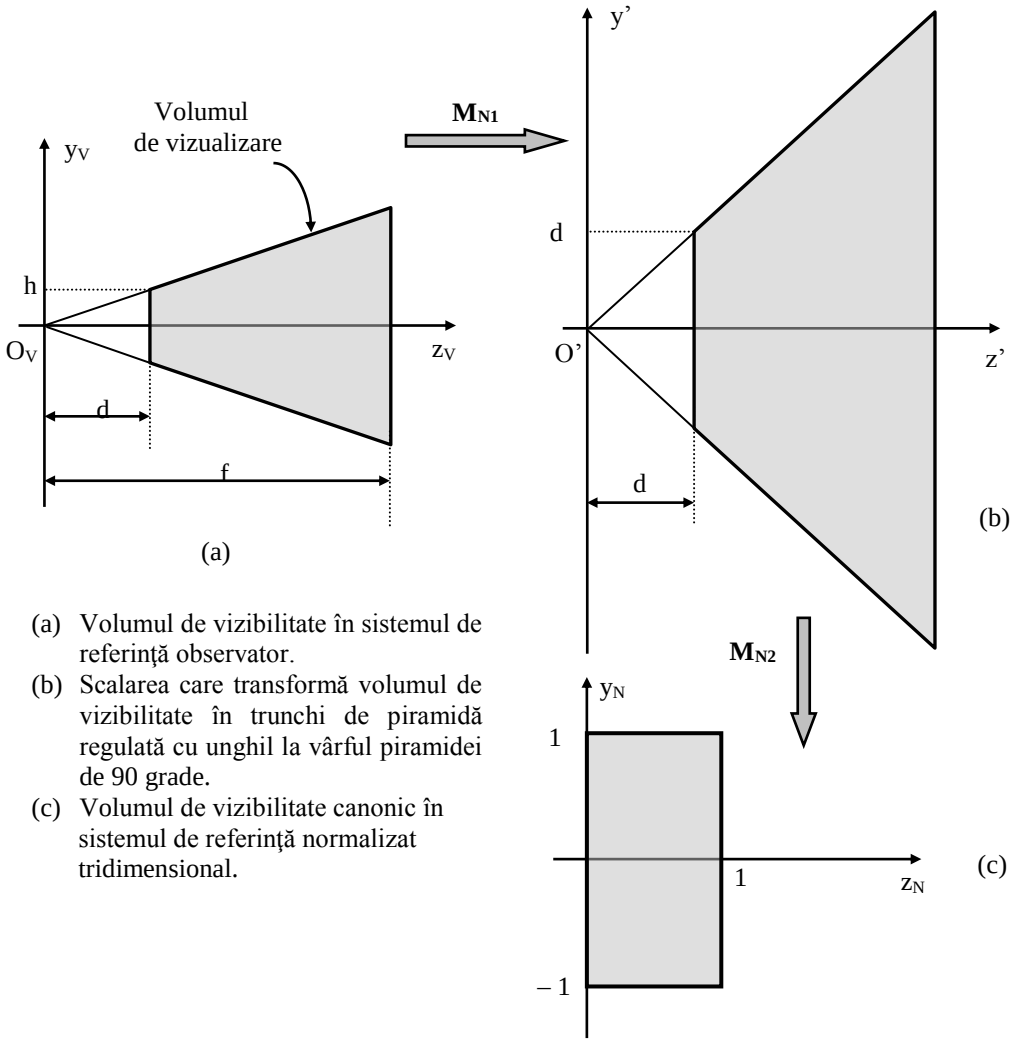
Matricea de transformare de normalizare $\mathbf{M_N}$ poate fi exprimată ca un produs de două matrice astfel:

$$\mathbf{M_N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & f/(f-d) & -df/(f-d) \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} d/g & 0 & 0 & 0 \\ 0 & d/h & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \mathbf{M_{N2}} \mathbf{M_{N1}}$$

Această reprezentare permite analiza mai detaliată a transformării de normalizare. Sistemul de referinţă normalizat astfel definit este un sistem de referinţă stâng. Matricea $\mathbf{M_{N1}}$ este o matrice de scalare neuniformă prin care trunchiul de piramidă de vizualizare este transformat într-un trunchi de piramidă regulată cu

unghiul la vârful piramidei de 90°. Matricea $\mathbf{M_{N2}}$ transformă acest trunchi de piramidă regulată în volumul canonic, care este un paralelipiped dreptunghic (Fig. 4.6).

În sistemul de referinţă normalizat se execută operaţia de decupare a obiectelor în coordonate normalizate omogene. În urma acestei operaţii, din totalitatea obiectelor scenei virtuale sunt reţinute numai acele obiecte sau părţi (rezultate prin decupare) aflate în interiorul volumului de vizualizare canonic, ceea ce înseamnă că, înainte de transformarea de normalizare, acestea se aflau în interiorul trunchiului de piramidă de vizualizare.



(a) Volumul de vizibilitate în sistemul de referinţă observator.
(b) Scalarea care transformă volumul de vizibilitate în trunchi de piramidă regulată cu unghil la vârful piramidei de 90 grade.
(c) Volumul de vizibilitate canonic în sistemul de referinţă normalizat tridimensional.

Fig. 4.6 Etapele transformării de normalizare.

Din coordonatele omogene X_N, Y_N, Z_N , w_N se pot deduce coordonatele x_N, y_N şi z_N ale punctului în sistemul de referinţă normalizat tridimensional:

$$x_N = X_N / w_N = \frac{d\, x_V}{g\, z_V};\ y_N = Y_N / w_N = \frac{d\, y_V}{h\, z_V};\ z_N = Z_N / w_N = \frac{f}{f-d}\left(1 - \frac{d}{z_V}\right)$$

S-au obţinut, evident, expresii identice cu cele din relaţiile 4.10, de la care s-a pornit pentru definirea transformării de proiecţie perspectivă.

Coordonatele x_N şi y_N în sistemul de referinţă normalizat reprezintă coordonatele proiecţiei ortografice în planul $z_N = 0$ (care este planul de vizualizare) a punctului corespunzător. Este de remarcat faptul că transformarea de proiecţie perspectivă din sistemul de referinţă observator a devenit o transformare de proiecţie ortografică în sistemul de referinţă normalizat. Dar aceaste operaţii de trecere de la coordonate omogene la coordonate tridimensionale şi proiecţia ortografică nu se efectuează în sistemul normalizat ci într-un alt sistem de referinţă, sistemul ecran 3D, obţinut prin transformarea fereastrei de vizualizare în poartă de afişare.

Unghiul dintre planele laterale ale piramidei de vizualizare (care trec prin axa $O_V y_V$ şi au ecuaţiile $x_V = \pm g\, z_V/d$) se numeşte unghi de vizualizare pe orizontală (*horizontal field of view– fovx*). Unghiul dintre planele piramidei de vizualizare care trec prin axa $O_V x_V$ se numeşte unghi de vizualizare pe verticală (*vertical field of view – fovy*). Valorile acestora se deduc din parametrii de proiecţie perspectivă:

$$fovx = 2\,\mathrm{arctg}\frac{g}{d},\ fovy = 2\,\mathrm{arctg}\frac{h}{d} \qquad (4.13)$$

Valorile unghiurilor de vizualizare variază pentru diferite sisteme grafice în funcţie de dispozitivele de afişare folosite.

O altă convenţie de definire a unui sistem de vizualizare este aceea de a considera sistemul de referinţă de observare ca un sistem drept, iar direcţia de observare şi volumul de vizualizare îndreptate către z_V negativ. Urmărind un raţionament asemănător celui de mai sus, se obţine aceeaşi expresie a matricii de normalizare (relaţia 4.12), în care valorile d şi f au valori negative, iar sistemul de referinţă normalizat rezultat este un sistem de referinţă stâng.

ν Exemplul 4.2

Se reia scena virtuală din Exemplul 3.3. Imaginile din figura de mai jos corespund obiectelor instanţiate în scena virtuală. Axele de coordonate reprezentate sunt axele sistemului de referinţă universal. Sistemul de referinţă de observare este un sistem de referinţă drept, iar direcţia de observare îndreptată către z_V negativ. Originea O_V are valori diferite în diferitele imagini ale figurii, aşa cum este indicat pentru fiecare dintre ele, şi orientare dată prin unghiurile de giraţie, tangaj, ruliu (χ ,θ, ρ).

Transformarea de perspectivă este definită prin parametrii: d = –1, f = –40, g = 1, h = 1. Succesiunea de transformări efectuate pentru obţinerea acestor imagini este:

(a) Transformarea de modelare, diferită pentru fiecare din cuburi, cu matricele $\mathbf{M_1}$, $\mathbf{M_2}$, $\mathbf{M_3}$, $\mathbf{M_4}$ calculate în Exemplul 3.3. După transformarea de modelare (instanţiere), toate obiectele sunt reprezentate în sistemul de referinţă universal şi, în continuare, vor fi supuse toate aceloraşi transformări.

(b) Transformarea de observare cu matricea:

$\mathbf{M_V} = \mathbf{R_Z}(-\rho)\mathbf{R_X}(-\theta)\mathbf{R_Y}(-\chi)\mathbf{T}(-x_V, -y_V, -z_V)$,

cu valorile parametrilor x_V, y_V, z_V, χ, θ, ρ corespunzătoare fiecărei imagini redate.

(c) Transformarea de normalizare cu matricea:

$$\mathbf{M_N} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1.025 & -1.025 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

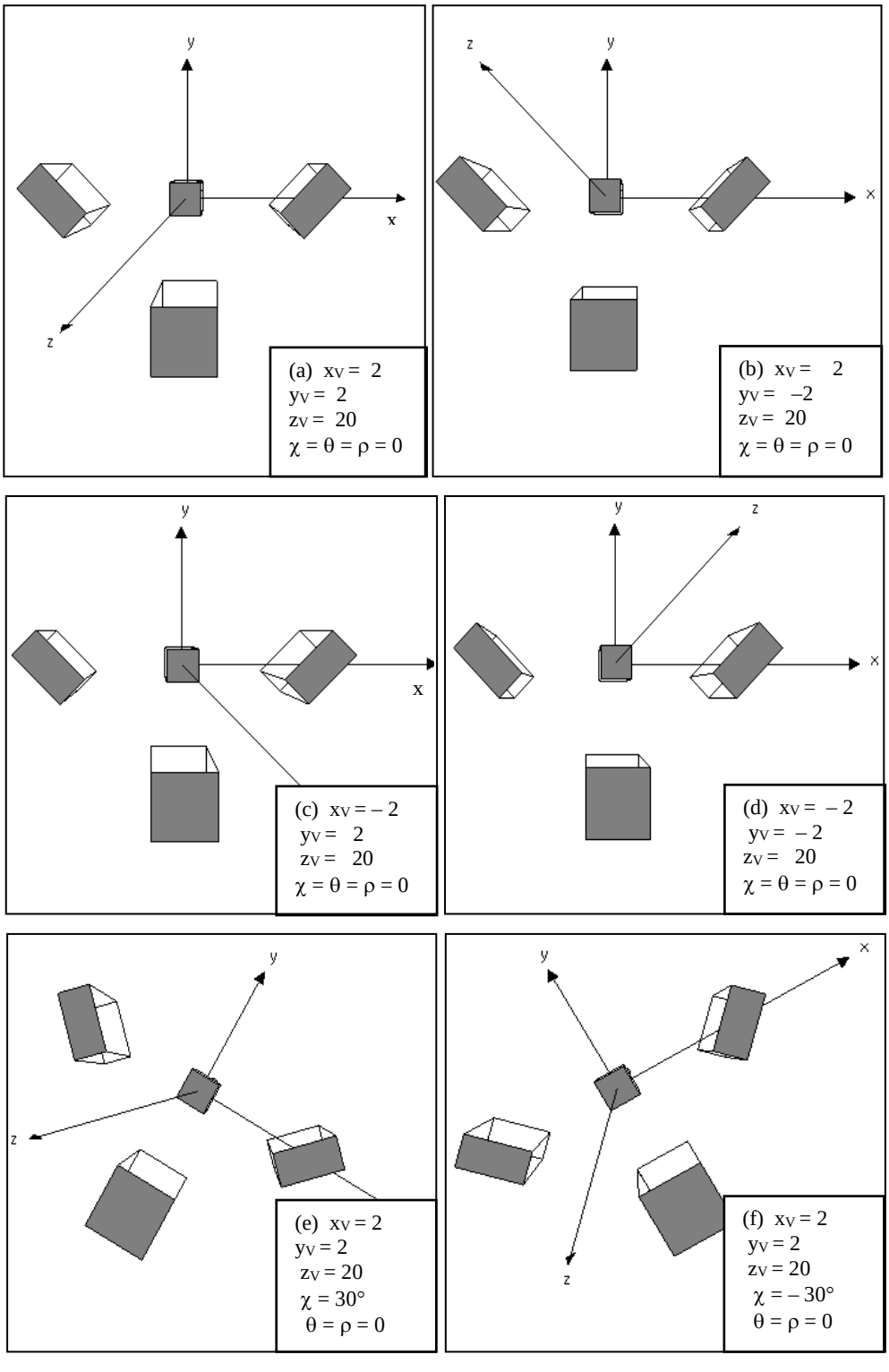

y
x
z
(a) $x_V = 2$ $y_V = 2$ $z_V = 20$ $\chi = \theta = \rho = 0$
z
y
x
(b) $x_V = 2$ $y_V = -2$ $z_V = 20$ $\chi = \theta = \rho = 0$
y
x
(c) $x_V = -2$ $y_V = 2$ $z_V = 20$ $\chi = \theta = \rho = 0$
y
z
x
(d) $x_V = -2$ $y_V = -2$ $z_V = 20$ $\chi = \theta = \rho = 0$
y
z
(e) $x_V = 2$ $y_V = 2$ $z_V = 20$ $\chi = 30°$ $\theta = \rho = 0$
y
x
z
(f) $x_V = 2$ $y_V = 2$ $z_V = 20$ $\chi = -30°$ $\theta = \rho = 0$

Fig. 4.7 Transformări de proiecţie perspectivă
pentru diferite puncte şi direcţii de observare a scenei

Imaginile generate pe ecran mai necesită şi alte transformări grafice, pe lângă cele prezentate până acum: transformarea în coordonate ecran 3D, prin care se efectuează o corespondenţă între fereastra de vizualizare şi zona de afişare alocată pe ecran (numită poartă –*viewport*), precum şi o transformare de rastru, prin care se generează mulţimea pixelilor afişaţi pe ecran. Aceste transformări vor fi descrise paragrafele următoare.

Imaginile prezentate au fost captate din fereastra de afişare a programului care execută toate operaţiile prezentate mai sus. Acest program a fost scris folosind biblioteca grafică OpenGL, care oferă funcţii adecvate acestor operaţii de transformare şi redare a imaginilor. Funcţiile OpenGL pentru realizarea transformărilor grafice vor fi descrise în secţiunea 6. ν

În acest sistem de proiecţie perspectivă s-a presupus că planul de vizualizare şi planul de vizibilitate apropiată coincid. Deducerea matricii de normalizare pentru cazul general, în care cele două plane sunt diferite, este lăsată ca un exerciţiu pentru cursanţi. De asemenea, tot ca un exerciţiu, se poate deduce matricea de normalizare în situaţia în care centrul ferestrei de vizualizare nu se află pe axa z_V.

Prin specificarea sistemului de referinţă de observare, a volumului de vizualizare şi a sistemului de proiecţie (tipul de proiecţie, centrul de proiecţie, direcţia de observare, fereastra de vizualizare) s-a definit un sistem de vizualizare care poate fi folosit pentru redarea obiectelor şi a scenelor tridimensionale. Acest sistem este destul de flexibil şi poate acoperi un număr mare de aplicaţii de grafică proiectivă (CAD-Computer Aided Design) şi unele aplicaţii de realitate virtuală neimersivă de tip desktop, în care mediul grafic este generat pe monitorul unui calculator.

Dar restricţiile impuse acestui sistem de vizualizare (de exemplu, s-a considerat că centrul de proiecţie coincide cu centrul sistemului de referinţă de observare) nu permit utilizarea lui în sisteme de realitate virtuală imersive sau semi-imersive în care este necesar un câmp de vizualizare de dimensiune mare. Astfel de situaţii se pot defini în sistemul de vizualizare standard PHIGS.

Sistemul de vizualizare standard

Imaginea generată pe un monitor sau proiectată de un proiector pe un ecran poate acoperi un câmp de vizualizare tipic de 35-45° vertical-orizontal. Pentru câmpuri de vizualizare mai mari se folosesc imagini multicanal, în care fiecare canal reprezintă un

volum de vizualizare care se proiectează în fereastra de vizualizare a acestuia. Mai multe canale de imagine, care generează imagini pe monitoare juxtapuse sau proiectate pe un ecran de dimensiuni corespunzătoare, permit obţinerea unor cîmpuri de vizualizare extinse în sistemele de realitate virtuală imersive.

Pentru generarea imaginilor multicanal se utilizează un sistem de vizualizare care permite definirea tuturor parametrilor necesari pentru generarea simultană a mai multor ferestre de vizualizare. Un astfel de sistem de vizualizare este cel definit în standardele GKS şi PHIGS, şi va fi descris în continuare.

În sistemul de vizualizare standard PHIGS, sistemul de referinţă observator (numit *view reference coordinate* –VRC) este definit prin punctul de observare VRP (*view reference point*), iar centrul de proiecţie este diferit de acesta şi este specificat prin punctul PRP (*projection reference point*). Planul de vizualizare (pe care se execută proiecţia imaginii) nu este în mod necesar perpendicular pe linia care uneşte centrul de proiecţie cu centrul ferestrei de vizualizare, ceea ce permite realizarea proiecţiilor oblice. Fereastra de vizualizare este poziţionată în orice loc în planul de vizualizare, ceea se permite definirea şi redarea simultană a mai multor imagini ale scenei.

Sistemul de vizualizare PHIGS se specifică în trei etape. În prima etapă se defineşte sistemul de referinţă de observare; în cea de-a doua etapă se defineşte transformarea de normalizare, iar în ultima etapă se defineşte transformarea fereastră-poartă.

3.1.7 Definirea sistemului de referinţă de observare

Sistemul de referinţă de observare se defineşte prin următorii parametri specificaţi în sistemul de referinţă universal:

- Punctul de observare VRP (*View Reference Point*).
- Direcţia normală la planul de vizualizare, VPN (*View Plane Normal*).
- Un vector a cărui proiecţie determină sensul de prezentare a imaginii, VUP (*view up vector*), şi care nu poate fi paralel cu vectorul VPN.

Punctul VRP, împreună cu vectorii VPN şi VUV definesc sistemul de referinţă de observare ca un sistem orientat după regula mâinii drepte, cu axele notate U, V, N. N este vectorul VPN, normal la planul de vizualizare, iar UV este un plan paralel cu planul de vizualizare (Fig. 4.8). Direcţia de observare este orientată în sensul negativ al vectorului VPN.

În planul de vizualizare se defineşte fereastra de vizualizare, care este un dreptunghi orientat cu laturile paralele cu axele u şi v ale unui sistem de referinţă bidimensional definit în acest plan. Acest sistem de referinţă are originea în punctul în care se proiectează (paralel cu direcţia VPN) centrul sistemului de referinţă de observare (VRP) pe planul de vizualizare. Direcţia axei v este paralelă cu proiecţia (după direcţia VPN) a vectorului VUV pe planul de vizualizare, astfel încât vectorul VUV specifică dacă scena este redată în direcţie verticală (în sus sau în jos) sau pe direcţie orizontală (spre dreapta sau spre stânga).

3.1.8 Definirea transformării de normalizare

Parametrii transformării de normalizare definesc corespondenţa între volumul de vizualizare specificat în sistemul de referinţă observator şi volumul canonic în sistemul de referinţă normalizat. Se specifică următorii parametri de definire a sistemului de referinţă normalizat relativ la sistemul de referinţă de observare:

- Tipul proiecţiei (paralelă sau perspectivă).
- Centrul de proiecţie PRP (*Projection Reference Point*).
- Distanţa VPD (*View Plane Distance*) a planului de vizualizare faţă de centrul sistemului de referinţă de observare, VRP.
- Distanţa planului apropiat (*near*) şi al planului depărtat de vizibilitate (*far*) faţă de centrul sistemului de referinţă de observare.
- Limitele ferestrei de vizualizare, u_{min}, u_{max}, v_{min}, v_{max}.

Planul de vizualizare este un plan paralel cu planul UV, aflat la distanţa dată de valoarea VPD faţă de centrul sistemului de observare, VRP. În planul de vizualizare se defineşte sistemul de coordonate plan uv, în care fereastra de vizualizare este un dreptunghi între limitele u_{min}, u_{max}, v_{min}, v_{max}.Volumul de vizualizare se defineşte prin două plane paralele cu planul de vizualizare aflate la distanţele *near* şi *far* de centrul sistemului de referinţă observator VRP şi planele care trec prin centrul de proiecţie PRP şi laturile ferestrei de vizualizare.

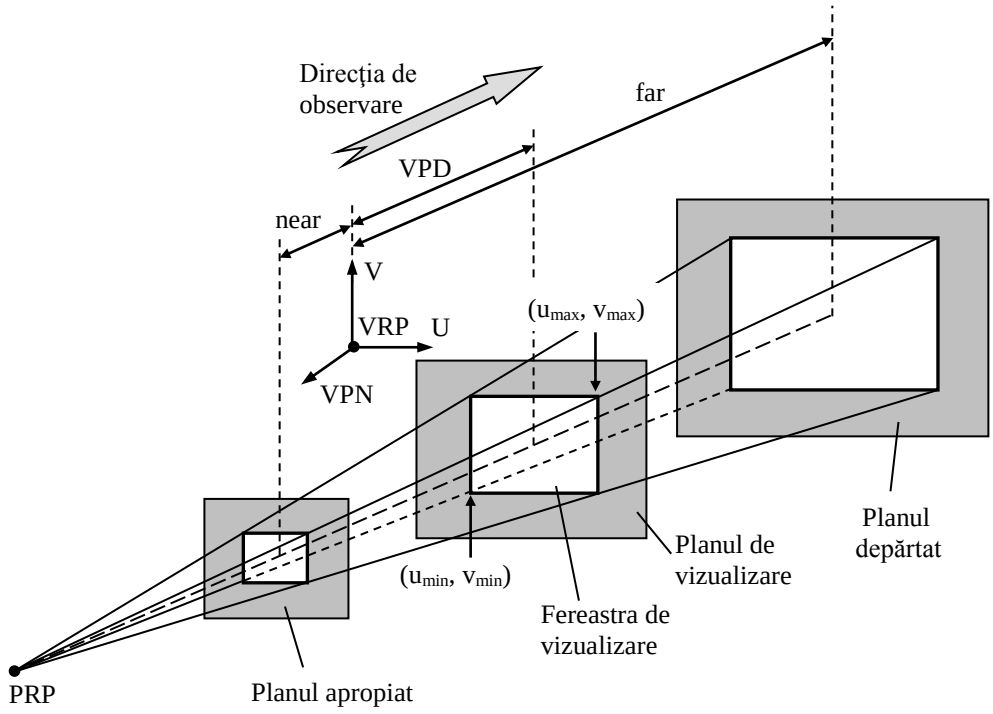


Fig. 4.8 Specificarea sistemului de vizualizare PHIGS în proiecţa perspectivă.

Există două posibilităţi de poziţionare a centrului de proiecţie PRP faţă de fereastra de vizualizare:

(a) Dacă linia care uneşte centrul de proiecţie cu centrul ferestrei de vizualizare este paralelă cu vectorul VPN, normal la planul de vizualizare, atunci se obţine o proiecţie perspectivă normală, echivalentă celei descrise anterior (Fig. 4.9(a)).
(b) Dacă punctul PRP nu se află pe linia paralelă cu vectorul VPN care trece prin centrul ferestrei de vizualizare, atunci proiecţia este o proiecţie oblică (Fig. 4.9(b)).

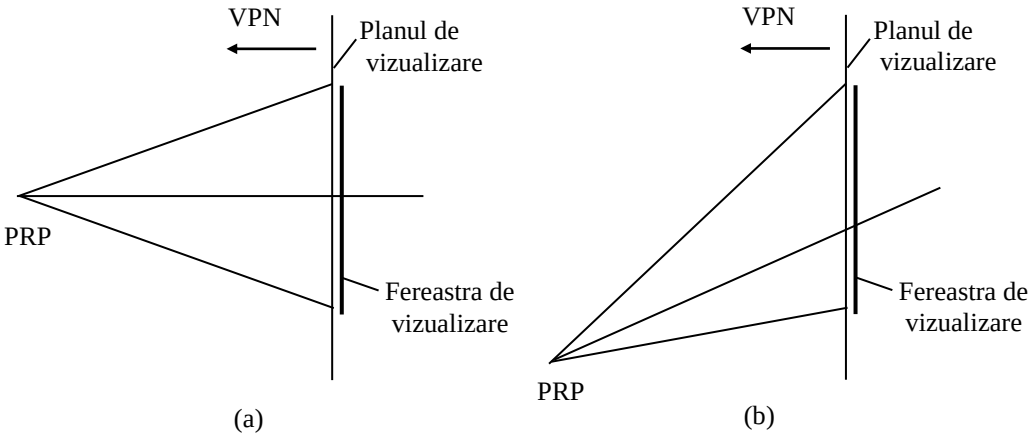


Fig. 4.9 (a) Proiecţia normală (b) Proiecţie oblică.

Implementarea sistemului de vizualizare PHIGS se face prin transformări geometrice succesive, la fel ca în exemplele prezentate până acum. Trebuie precizat că în sistemul PHIGS standard sistemul de referinţă de observare UVN este sistem drept, iar volumul de vizualizare şi direcţia de observare sunt orientate către axa – N.

O primă transformare geometrică este o schimbare de sistem de referinţă din sistemul de referinţă de observare UVN într-un sistem de referinţă notat Oxyz, cu centrul în centrul de proiecţie PRP şi cu axele paralele cu axele sistemului UVN (s-a ales această denumire pentru simplificarea notaţiilor, dar acest sistem nu este sistemul de referinţă universal pentru care, în mod obişnuit, s-a folosit notaţia Oxyz). Această transformare se efectuează printr-o translaţie inversă celei definite de vectorul de poziţie al punctului PRP.

Ca urmare a acestei transformări se modifică corespunzător valorile VPD, near, far, u_{min}, u_{max}, v_{min}, v_{max}. Fie d, n, f, x_{min}, x_{max}, y_{min}, respectiv y_{max}, valorile rezultate după translaţie. Aceste valori sunt numere reale şi pot fi pozitive sau negative. Matricea de transformare de normalizare pornind din acest sistem Oxyz, se poate descompune în produs de trei matrice:

$$\mathbf{M_N = M_{N2}\, M_{N1} M_{N0}}$$

Matricele $\mathbf{M_{N1}}$ şi $\mathbf{M_{N2}}$ au semnificaţii asemănătoare cu matricele dezvoltate pentru sistemul de vizualizare precedent. Cealaltă transformare componentă, $\mathbf{M_{N0}}$, este o transformare de forfecare, prin care linia centrală a volumului de vizualizare (cea care

unește centrul de proiecție PRP cu centrul ferestrei de vizualizare, Fig. 4.10) se suprapune cu aza z a sistemului.

Transformarea de forfecare modifică valorile coordonatelor x şi y cu valori proporționale cu z.

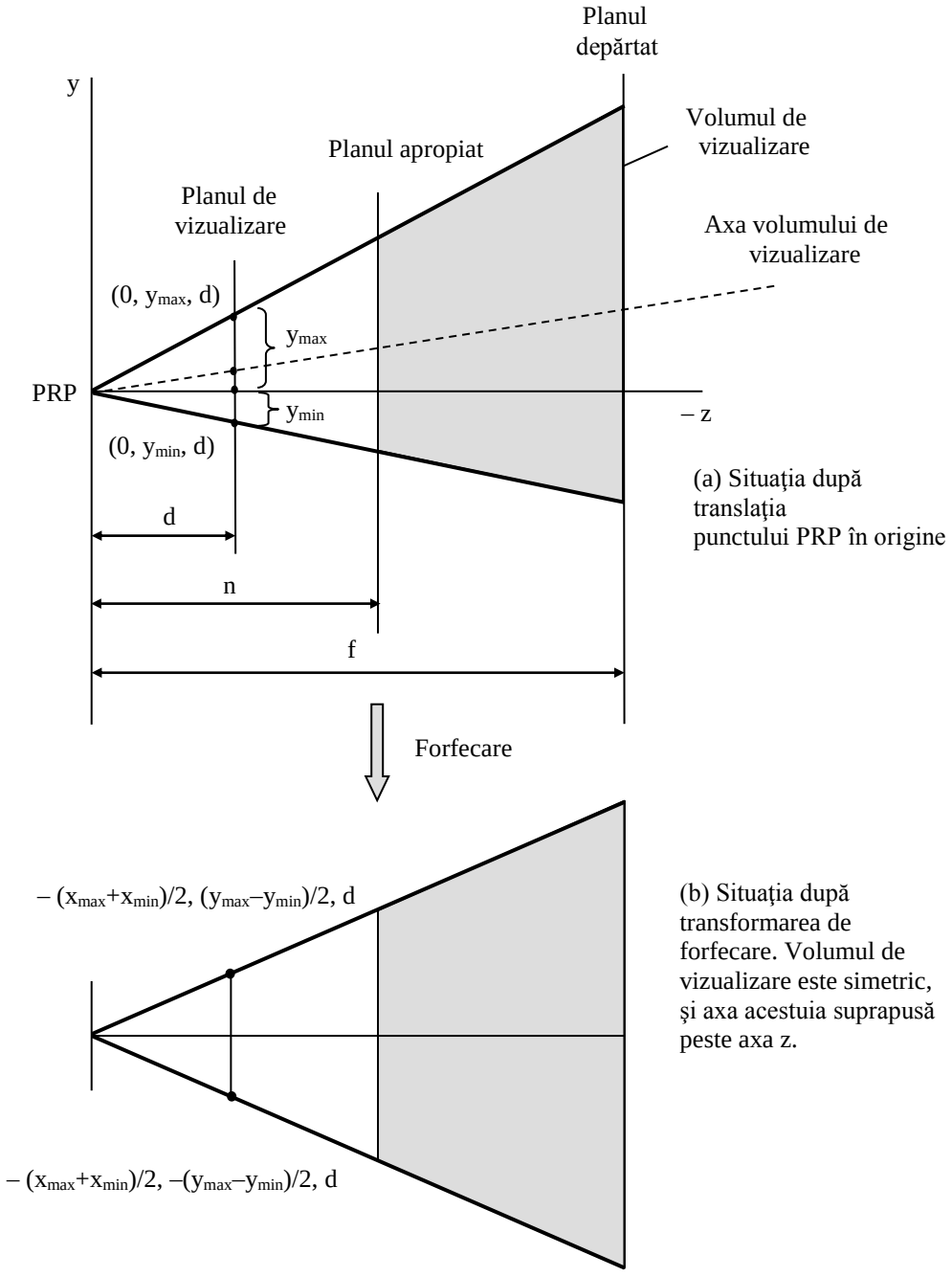


Fig. 4.10 Transformare de forfecare în sistemul de vizualizare standard.

Este simplu de observat că matricea:

$$\mathbf{M_{N0}} = \begin{bmatrix} 1 & 0 & -(x_{max}+x_{min})/2d & 0 \\ 0 & 1 & -(y_{max}+y_{min})/2d & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.14}$$

efectuează o simetrizare a volumului de vizualizare. De exemplu, intersecţiile muchiilor orizontale ale ferestrei de vizualizare cu planul x = 0 se transformă astfel:

$$\mathbf{M_{N0}}[0 \quad y_{max} \quad d \quad 1]^T = [-(x_{max}+x_{min})/2 \quad (y_{max}-y_{min})/2 \quad d \quad 1]^T$$

$$\mathbf{M_{N0}}[0 \quad y_{min} \quad d \quad 1]^T = [-(x_{max}+x_{min})/2 \quad -(y_{max}-y_{min})/2 \quad d \quad 1]^T$$

A doua transformare componentă este transformarea de scalare $\mathbf{M_{N1}}$, prin care volumul de vizualizare este transformat într-un volum simetric, cu unghiul de la vârful piramidei de 90°:

$$\mathbf{M_{N1}} = \begin{bmatrix} 2d/(x_{max}-x_{min}) & 0 & 0 & 0 \\ 0 & 2d/(y_{max}-y_{min}) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.15}$$

De exemplu, transformarea compusă $\mathbf{M_{N1}M_{N0}}$ aplicată punctului din centrul ferestrei de vizualizare, îl transformă într-un punct aflat pe axa z negativă (d este negativ):

$$\mathbf{M_{N1}M_{N0}}[(x_{max}+x_{min})/2 \quad (y_{max}+y_{min})/2 \quad d \quad 1]^T = [0 \quad 0 \quad d \quad 1]^T$$

Ultima componentă a transformării de normalizare transformă volumul de vizualizare în volum canonic (paralelipiped dreptunghic). Matricea acestei transformări, $\mathbf{M_{N2}}$, este asemănătoare matricei descrise în paragraful precedent, dar se ţine seama de faptul că planul de vizibilitate apropiat nu mai coincide cu planul de vizualizare:

$$\mathbf{M_{N2}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & f/(f-n) & -nf/(f-n) \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.16}$$

Sistemul normalizat astfel definit este un sistem de referinţă stâng. În acest sistem se efectuează operaţiile de decupare a obiectelor, după care secvenţa transformărilor necesare pentru generarea imaginii continuă cu transformarea în sistemul de referinţă ecran 3D.

Sistemul de referinţă ecran 3D

Transformarea în sistemul de referinţă ecran 3D efectuează o corespondenţă între fereastra din planul de vizualizare şi zona de afişare alocată pe display numită poartă de afişare (*viewport*). Fereastra de vizualizare este definită într-un sistem de referinţă bidimensional în planul $z_N = 0$ al sistemului de referinţă normalizat. Poarta de afişare este definită într-un sistem de referinţă bidimensional în planul $z_S = 0$ al sistemului de referinţă ecran 3D şi este, în general, o zonă rectangulară de dimensiuni egale cu numărul de pixeli corespunzători pe orizontală şi pe verticală a zonei afişate. (Fig. 4.11). Transformarea din sistemul de referinţă normalizat în sistemul ecran 3D lasă coordonata z nemodificată. Sistemul de referinţă ecran 3D astfel definit este un sistem de referinţă stâng, la fel ca şi sistemul de referinţă normalizat.

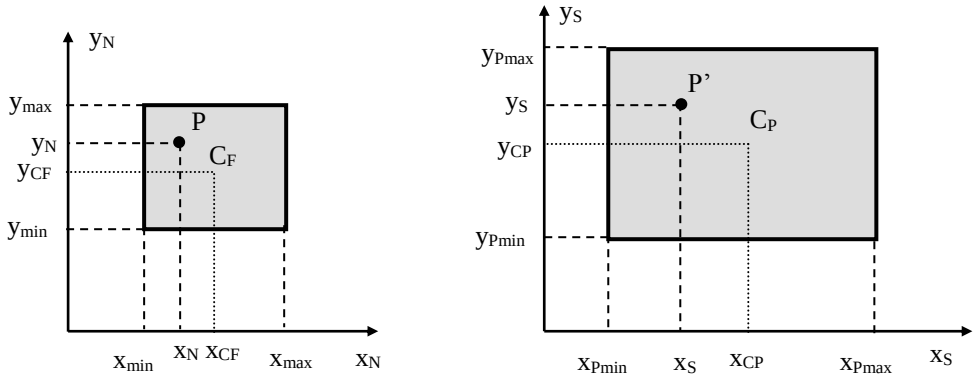


Fig. 4.11 Transformarea de la fereastra de vizualizare la poarta de afişare.

Pentru definirea transformării în sistemul de referinţă ecran 3D, se calculează mai întâi coordonatele centrului C_F al ferestrei şi ale centrului C_P al porţii:

$$\begin{aligned} x_{CF} &= (x_{max} + x_{min})/2;\ y_{CF} = (y_{max} + y_{min})/2 \\ x_{CP} &= (x_{Pmax} + x_{Pmin})/2;\ y_{CP} = (y_{Pmax} + y_{Pmin})/2 \end{aligned} \tag{4.17}$$

Corespondenţa dintre coordonatele unui punct $P(x_N, y_N)$ din fereastră cu ale punctului $P'(x_S, y_S)$ în poarta de afişare se formulează matematic astfel:

$$\frac{x_S - x_{CP}}{x_N - x_{CF}} = \frac{x_{Pmax} - x_{Pmin}}{x_{max} - x_{min}}; \frac{y_S - y_{CP}}{y_S - y_{CF}} = \frac{y_{Pmax} - y_{Pmin}}{y_{max} - y_{min}}$$

Se definesc factorii de scalare ai transformării:

$$s_x = \frac{x_{Pmax} - x_{Pmin}}{x_{max} - x_{min}}$$
$$s_y = \frac{y_{Pmax} - y_{Pmin}}{y_{max} - y_{min}} \qquad (4.18)$$
$$s_z = 1$$

Rezultă:

$$x_S = x_{CP} + s_x(x_N - x_{CF}) = s_x x_N + x_{CP} - s_x x_{CF}$$
$$y_S = y_{CP} + s_y(y - y_{CF}) = s_y y_N + y_{CP} - s_y y_{CF} \qquad (4.19)$$
$$z_S = z_N$$

Se poate deduce matricea de transformare ecran 3D prin compunerea a trei transformări succesive. Mai întâi se execută o translaţie, astfel ca centrul C_F al ferestrei să ajungă în origine, deci cu matricea de translaţie $\mathbf{T}(-x_{CF}, -y_{CF}, 0)$.

Fereastra centrată în origine se scalează cu factorii de scară s_x, s_y, definiţi de relaţia 4.18, astfel încât fereastra să ajungă la dimensiunea porţii, iar coordonata z se lasă nemodificată. Matricea de scalare este deci $\mathbf{S}(s_x, s_y, s_z)$. Ultima transformare necesară este o translaţie, prin care centrul porţii este adus în punctul C_P, deci cu matricea de translaţie $\mathbf{T}(x_{CP}, y_{CP}, 0)$. Rezultă matricea de transformare din sistemul de referinţă normalizat în sistemul de referinţă ecran 3D:

$$\mathbf{M_{NS}} = \begin{bmatrix} 1 & 0 & 0 & x_{CP} \\ 0 & 1 & 0 & y_{CP} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_{CF} \\ 0 & 1 & 0 & -y_{CF} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{M_{NS}} = \begin{bmatrix} s_x & 0 & 0 & x_{CP} - s_x x_{CF} \\ 0 & s_y & 0 & y_{CP} - s_y y_{CF} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \quad \begin{bmatrix} X_S \\ Y_S \\ Z_S \\ w_S \end{bmatrix} = \mathbf{M_{NS}} \begin{bmatrix} X_N \\ Y_N \\ Z_N \\ w_N \end{bmatrix} \qquad (4.20)$$

Din aceste relaţii rezultă $Z_S = Z_N$ şi $w_S = w_N$. În sistemul de referinţă ecran 3D se trece de la coordonatele omogene X_S, Y_S, Z_S, w_S, la coordonatele tridimensionale x_S, y_S, z_S prin împărţirea cu w_S.

Planul $z_S = 0$ din sistemul de referinţă ecran 3D este planul de proiecţie. Coordonatele x_S şi y_S reprezintă coordonatele proiecţiei ortografice în planul $z_S = 0$ (care este planul de porţii de afişare). Transformarea de proiecţie perspectivă din sistemul de referinţă de observare a devenit o transformare de proiecţie ortografică în sistemul de referinţă ecran 3D. Coordonata z_S în sistemul de referinţă ecran 3D este utilizată în algoritmii de eliminare a suprafeţelor ascunse. Volumul de vizualizare în sistemul ecran 3D este un paralelipiped dreptunghic cu baza un dreptunghi de

dimensiuni egale cu dimensiunile porţii afişate şi înălţimea egală cu 1 ($z_S \in [0,1]$) (Fig. 4.12).

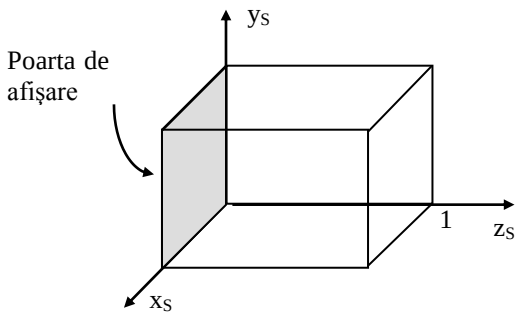


Fig. 4.12 Volumul de vizualizare în sistemul de referinţă ecran 3D

Decuparea obiectelor

Într-un sistem în care este definit punctul de observare şi un volum de vizualizare, nu interesează decât obiectele din interiorul acestui volum, a căror proiecţie se generează în fereastra de vizualizare. De aceea, în vizualizarea scenelor tridimensionale, se execută operaţia de decupare la volumul de vizualizare, prin care se reţin numai obiectele sau părţile din obiecte cuprinse în acest volum.

Operaţia de decupare la volumul de vizualizare poate fi definită astfel: fiind dat un obiect tridimensional şi un volum de vizualizare, pot exista trei posibilităţi:

- Obiectul este complet în interiorul volumului şi, în acest caz, el este proiectat în fereastră şi redat pe display.
- Obiectul este complet în afara volumului de vizualizare şi, în acest caz, el poate fi ignorat.
- Obiectul intersectează volumul de vizualizare; în acest caz se decupează obiectul şi partea vizibilă (cuprinsă în interiorul volumului) este proiectată în fereastra de vizualizare şi redată pe display.

Decuparea obiectelor tridimensionale la volumul de vizualizare se execută în sistemul de referinţă normalizat în coordonate omogene, înainte de împărţirea cu w.

Operaţia de decupare în spaţiul tridimensional reprezintă o extindere a decupării în planul bidimensional, de aceea se va prezenta mai întâi decuparea în plan.

Decuparea în plan

Fiind dat un poligon oarecare în plan şi o regiune de decupare, problema decupării poligonului este de a reţine numai acea parte din poligon care se află în interiorul regiunii de decupare. Se consideră cazul cel mai frecvent întâlnit, în care regiunea de decupare este un dreptunghi cu laturile paralele cu axele de coordonate. Această regiune de decupare corespunde ferestrei de vizualizare.

Majoritatea algoritmilor de decupare, atât în plan cât şi în spaţiu, se bazează pe algoritmul de decupare Sutherland-Hodgman. Algoritmul Sutherland-Hodgman împarte planul bidimensional în nouă regiuni, după poziţia faţă de regiunea dreptunghiulară de decupare dată prin ecuaţiile a patru drepte: $x = x_{min}$; $x = x_{max}$; $y = y_{min}$; $y = y_{max}$. (Fig. 4.13).

Fiecărei regiuni îi corespunde un cod de vizibilitate (cunoscut sub numele de cod Sutherland-Cohen) reprezentat pe 4 biţi, b_0, b_1, b_2, b_3, definiţi astfel:

$b_0 = semn(x - x_{min})$
$b_1 = semn(x_{max} - x)$
$b_2 = semn(y - y_{min})$
$b_3 = semn(y_{max} - y)$

În continuare codurile de vizibilitate se reprezintă ca o secvenţă de cifre binare.

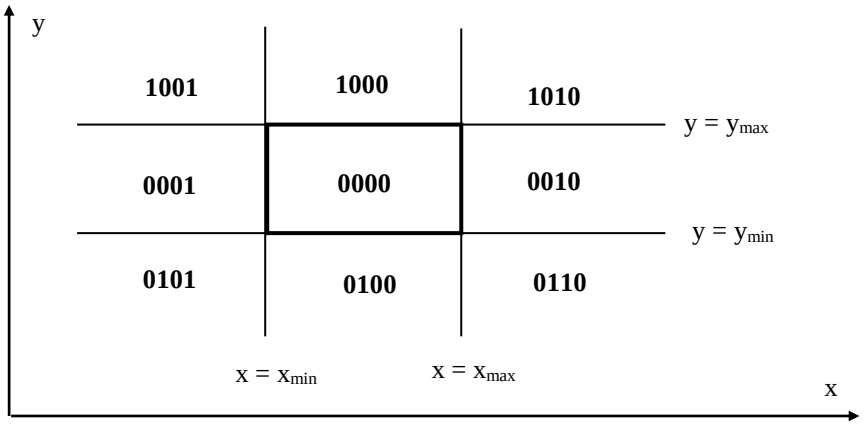


Fig. 4.13 Codurile de vizibilitate Sutherland-Cohen.

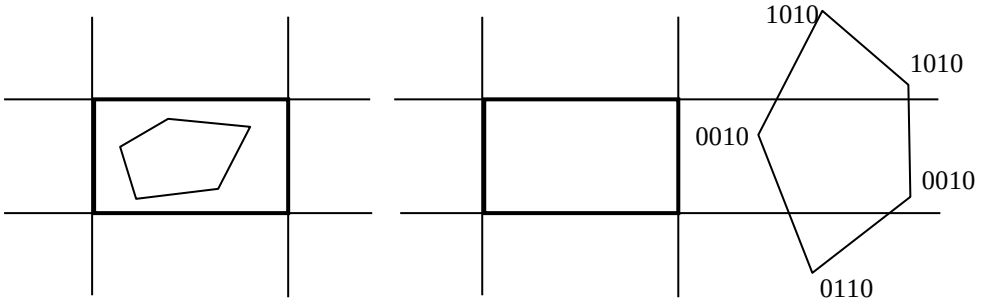
Un poligon oarecare din plan poate fi complet inclus în regiunea de decupare, poate fi complet în afara regiunii de decupare sau o poate intersecta. Folosind codurile de vizibilitate ale vârfurilor poligonului se pot identifica direct (fără să se execute calcule de intersecţii) două situaţii ale poligonului faţă de regiunea de decupare:

- *Acceptare sigură*. Dacă toate vârfurile poligonului se află în regiunea de decupare, atunci polinomul este vizibil în întregime. Fie c1, c2, …., cn, codurile de vizibilitate ale celor n vârfuri ale poligonului şi SC rezultatul reuniunii acestor coduri (operaţia OR inclusiv): SC = c1| c2 |…| cn. Dacă SC = 0, atunci poligonul este sigur vizibil (Fig. 4.14 (a)).
- *Rejecţie sigură*. Dacă toate vârfurile unui poligon se află într-unul din semiplanele invizibile ($x < x_{min}$; $x > x_{max}$; $y < y_{min}$; $y > y_{max}$.), atunci tot poligonul este sigur invizibil. Pentru efectuarea testului de rejecţie sigură se calculează intersecţia codurilor de vizibilitate (operaţia AND): PC = c1 & c2 &…& cn. Dacă PC are valoare diferită de 0, atunci poligonul este sigur invizibil deoarece toate punctele se află în semiplanul invizibil corespunzător unui bit diferit de 0 din codul PC (Fig. 4.14 (b)).

Dacă nu este îndeplinită nici condiţia de acceptare sigură, nici condiţia de rejecţie sigură, atunci decuparea se calculează prin intersecţia laturilor poligonului cu dreptele care mărginesc regiunea de decupare (Fig. 4.14 (c)).

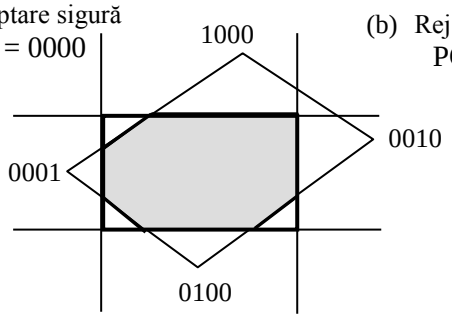
Dreapta (infinită) corespunzătoare unei laturi a regiunii de decupare împarte planul în două semiplane: semiplanul (regiunea) vizibil, care se află de aceeaşi parte a dreptei ca şi regiunea de decupare, şi semiplanul invizibil, care se află de partea opusă faţă de regiunea de decupare. De exemplu, pentru latura $x = x_{min}$, semiplanul $x < x_{min}$, este invizibil iar semiplanul $x \geq x_{min}$ este vizibil.

În algoritmul de decupare Sutherland-Hodgman, decuparea unui poligon faţă de o regiune dată se efectuează într-un număr de paşi egal cu numărul de laturi ale regiunii de decupare. În fiecare pas se decupează poligonul (iniţial sau provenit dintr-un pas anterior) cu o latură a regiunii de decupare: se elimină acea porţiune din poligon care se află în semiplanul invizibil corespunzător laturii şi se reţine numai porţiunea vizibilă.



(a) Acceptare sigură
SC = 0000

(b) Rejecţie sigură
PC = 0010



(c) Decupare prin intersecţie
SC = 1111, PC = 0000

Fig. 4.14 Teste de vizibilitate pentru decuparea în plan.

Poligonul se reprezintă printr-o listă ordonată de vârfuri $V_1V_2\ldots V_n$, şi fiecare pereche de două vârfuri consecutive ($V_i V_{i+1}$), pentru $i < n$, şi (V_nV_1) reprezintă o latură a poligonului ca un segment de dreaptă orientat (vector).

În fiecare pas de decupare se porneşte de la un poligon dat printr-o listă ordonată de vârfuri şi se creează lista vârfurilor poligonului rezultat prin decupare relativ la dreapta suport a unei laturi a regiunii de decupare. Iniţial, lista vârfurilor poligonului rezultat este considerată vidă. Un segment orientat V_iV_j se poate afla într-una din patru situaţii posibile faţă de o latură de decupare (Fig. 4.15).

(a) Dacă ambele vârfuri se află în semiplanul vizibil corespunzător laturii de decupare, atunci se adaugă în listă vârful V_j (Fig. 4.15 (a)).

(b) Dacă ambele vârfuri se află în semiplanul invizibil, atunci nu se adaugă nici un vârf în listă (Fig. 4.15 (b)).

(c) Dacă primul vârf al segmentului (V_i) se află în semiplanul vizibil, iar al doilea (V_j) se află în semiplanul invizibil (segmentul este orientat de la semiplanul vizibil către semiplanul invizibil), atunci se calculează intersecţia I dintre segment şi dreapta de decupare şi se adaugă intersecţia I în listă (Fig. 4.15(c)).

(d) Dacă primul vârf al segmentului (V_i) se află în semiplanul invizibil, iar al doilea (V_j) se află în semiplanul vizibil (segmentul este orientat de la semiplanul invizibil către semiplanul vizibil), atunci se calculează intersecţia I dintre segment şi dreapta de decupare în lista vârfurilor se adaugă intersecţia I şi apoi vârful V_j (Fig. 4.15 (d)).

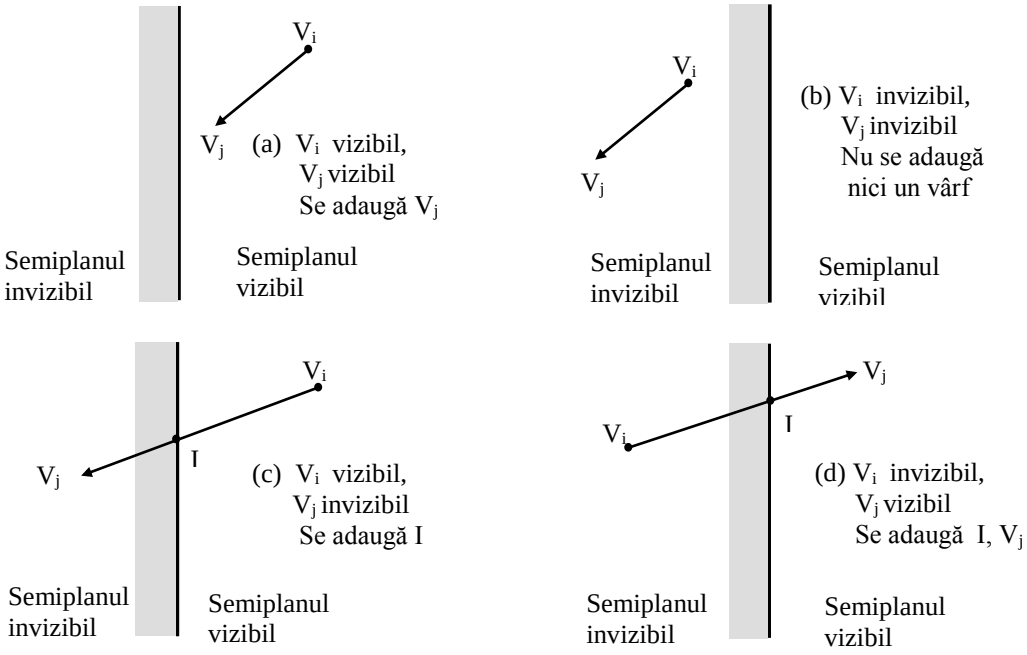
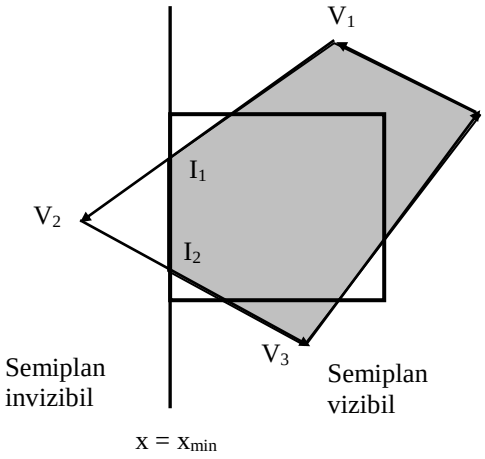
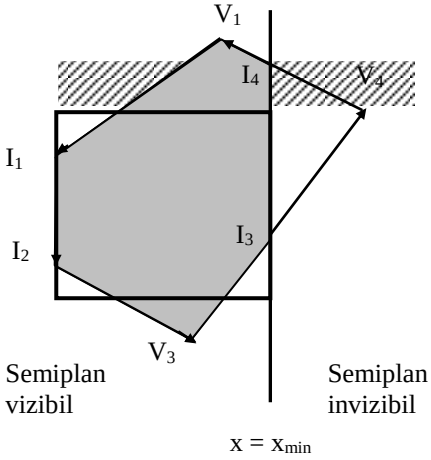


Fig. 4.15 Intersecţia dintre o latură a poligonului şi o dreaptă de decupare.

În primul pas, poligonul iniţial $V_1V_2\ldots V_n$ se decupează relativ la latura $x = x_{min}$ prin parcurgerea în ordine a tuturor segmentelor orientate ale poligonului (Fig. 4.16(a)). Poligonul rezultat se decupează relativ la următoarea latură a regiunii de decupare (Fig. 4.16 (b)). După parcurgerea unui număr de paşi egal cu numărul de laturi ale regiunii de decupare se obţine poligonul complet decupat (Fig. 4.16(d)).

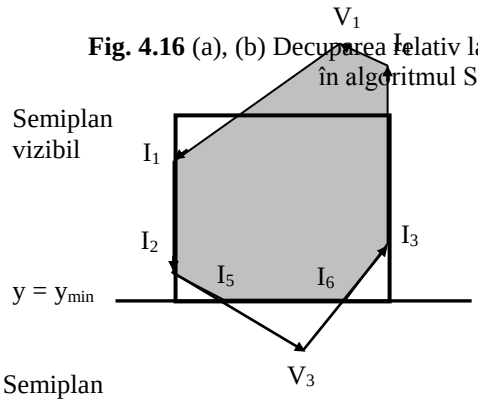


(a) Poligonul $V_1V_2V_3V_4$ decupat în semiplanul $x \geq x_{min}$

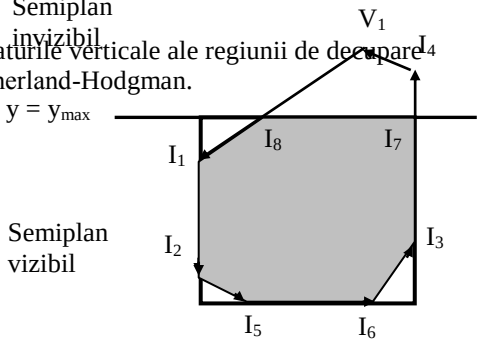


(b) Poligonul $I_1I_2V_3V_4V_1$ decupat în semiplanul $x \leq x_{max}$

Fig. 4.16 (a), (b) Decuparea relativ la laturile verticale ale regiunii de decupare în algoritmul Sutherland-Hodgman.



(c) Poligonul $I_2V_3\ I_3\ I_4V$ pat în semiplanul $y \geq$



(d) Poligonul $I_2I_5\ I_6\ I_3I_4V_1$ decupat în semiplanul $y \geq y_{min}$

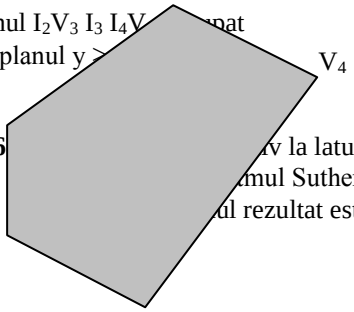


Fig. 4.16 iv la laturile orizontale ale regiunii de decupare
tmul Sutherland-Hodgman.
ul rezultat este: $I_5\ I_6\ I_3I_7\ I_8\ I_1\ I_2$

În Fig.4.16 este ilustrat algoritmul de decupare Sutherland-Hodgman al unui poligon convex relativ la o regiune de decupare rectangulară, dar acest algoritm poate fi folosit în cazul general, al decupării oricărui tip de poligon (convex, concav sau cu găuri) relativ la o regiune de decupare oarecare.

Decuparea (în plan sau în spaţiu) poate fi executată în orice operaţie care necesită construirea de poligoare sau obiecte tridimensionale noi prin calcul, pornind de la obiecte de bază, care sunt intersectate cu alte obiecte sau suprafeţe.
În generarea imaginilor bidimensionale, decuparea, ca operaţie componentă a secvenţei de transformări de vizualizare, se efectuează relativ la fereastra de vizualizare, care este o regiune rectangulară.
Poligoanele sau liniile rezultate, limitate la dimensiunea ferestrei, sunt transformate în sistemul de referinţă al porţii de afişare şi redate pe display. Dacă nu se efectuează decuparea poligoanelor la limitele ferestrei, coordonatele vârfurilor în sistemul de referinţă al porţii de afişare depăşesc limitele porţii, iar rezultatul conversiei de rastru este incorect.
Se poate efectua o decupare a poligoanelor şi liniilor şi în poarta de afişare, prin testul individual al coordonatelor fiecărui pixel şi rejectarea acelora care nu se încadrează în limitele porţii de afişare. Acesta este însă un procedeu deosebit de costisitor ca timp de execuţie şi nu se pune problema utilizării lui în grafica interactivă şi în realitatea virtuală.

Decuparea suprafeţelor relativ la volumul de vizualizare

Algoritmii de decupare a obiectelor în spaţiul tridimensional pot fi consideraţi ca o extensie a algoritmilor de decupare din plan, prin înlocuirea regiunii de decupare plane cu un volum de decupare şi efectuarea calculelor de intersecţie a unei suprafeţe în spaţiu cu planele volumului de decupare.
Decuparea la volumul de vizualizare este un caz particular al decupării spaţiale, în care volumul de decupare este volumul de vizualizare (trunchiul de piramidă de vizualizare) definit în sistemul de referinţă de observare.
În continuare se analizează modul de execuţie a decupării pentru sistemul de vizualizare definit în sectiunea precedentă (Fig. 4.5) şi se va evidenţia de ce decuparea se execută cel mai eficient în sistemul de referinţă normalizat, afirmaţie care nu a fost încă motivată.
Volumul de decupare se defineşte prin şase plane în sistemul de referinţă de observare, ale căror ecuaţii sunt date de relaţiile 4.9.
Fiecare plan împarte spaţiul tridimensional în două semispaţii, semispaţiul vizibil, aflat de aceeaşi parte a planului ca şi volumul de decupare, şi semispaţiul invizibil, în partea opusă a volumului. Decuparea unei suprafeţe în raport cu acest plan se efectuează asemănător cu decuparea unui poligon faţă de o latură de decupare: fiecare latură orientată introduce zero, unul sau două vârfuri în lista se vârfuri a suprafeţei rezultate, în funcţie de orientare şi de intersecţia ei cu planul. Un vârf este vizibil în raport cu un plan dacă se află în semispaţiul vizibil determinat de planul dat. De exemplu, un vârf este vizibil în raport cu planul

$y_V = hz_V / d$, dacă $y_V \le hz_V/d$ (Fig. 4.17 (a)).

În mod asemănător, decuparea relativ la celelate plane ale volumului de decupare necesită testele:

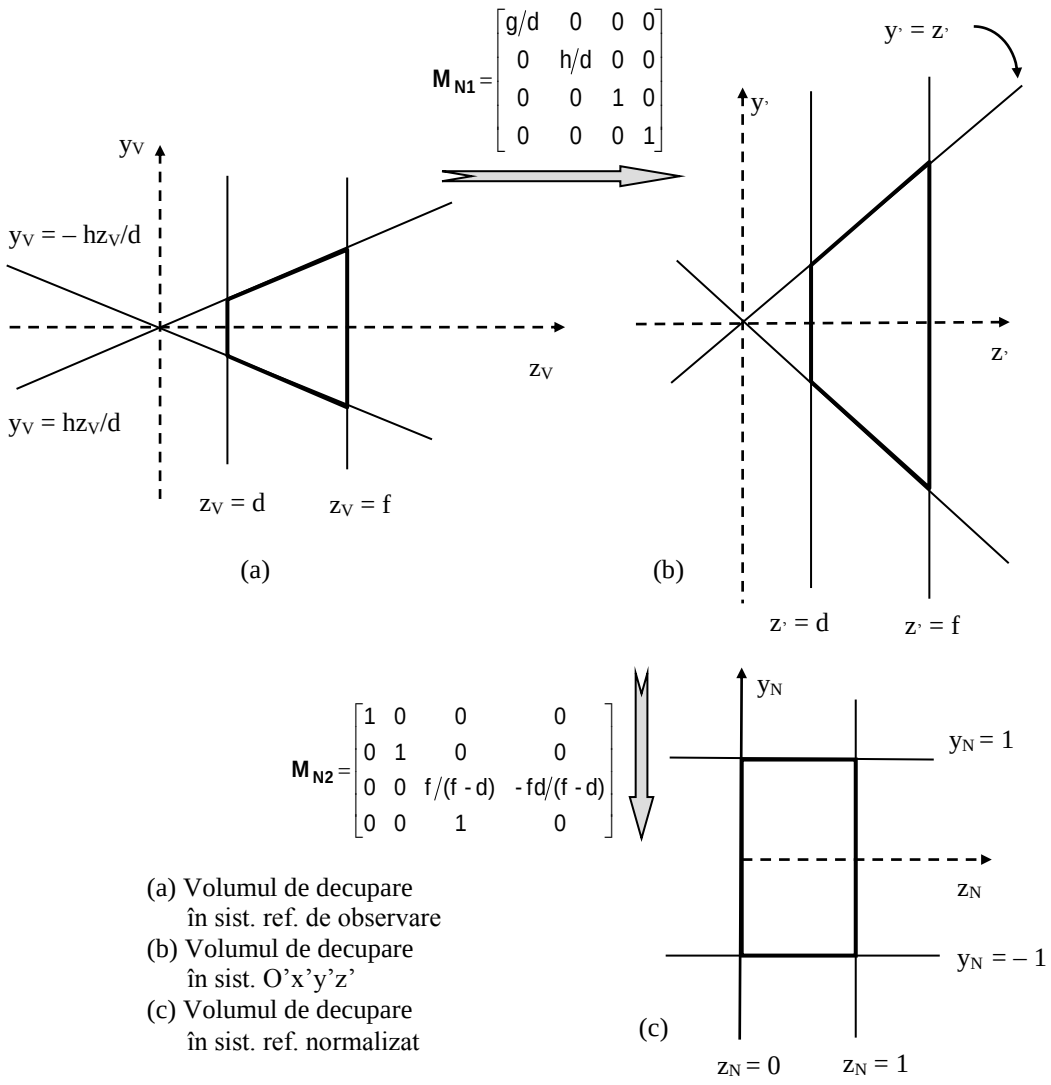
$$y_V \ge -\frac{h}{d} z_V;\ x_V \le \frac{g}{d} z_V;\ x_V \ge -\frac{g}{d} z_V;\ z_V \ge d;\ z_V \le f \qquad (4.21)$$

Aceste teste implică operaţii asupra coordonatelor vârfurilor, operaţii care în formă matriceală înseamnă înmulţirea cu matricea $\mathbf{M_{N1}}$, componentă a matricei de normalizare. Aşa cum se poate observa în Fig. 4.17(b), matricea $\mathbf{M_{N1}}$ transformă sistemul de referinţă de observare într-un sistem de referinţă O'x'y'z', în care volumul de vizualizare (deci volumul de decupare) este un trunchi de piramidă regulată cu unghiul

de la vârful piramidei de 90°. Relaţiile 4.21 de testare a vizibilităţii unui punct în sistemul de referinţă de observare sunt echivalente cu următoarele relaţii în sistemul O'x'y'z':

$$x' \le z';\ x' \ge -z';\ y' \le z';\ y' \ge -z';\ z' \le f;\ z' \ge d \qquad (4.22)$$

Dacă s-ar executa decuparea în sistemul de referinţă intermediar O'x'y'z', atunci toate punctele din sistemul de referinţă de observare se transformă mai întâi cu matricea $\mathbf{M_{N1}}$, pentru trecerea în sistemul O'x'y'z', apoi se execută decuparea, iar după decupare, vârfurile vizibile se transformă cu matricea $\mathbf{M_{N2}}$, pentru trecere în sistemul de referinţă normalizat şi proiecţie în fereastra de vizualizare (Fig. 4.17(c)). Aplicarea separată a două transformări, mai întâi prin matricea $\mathbf{M_{N1}}$ şi apoi prin matricea $\mathbf{M_{N2}}$ este costisitoarte şi nenecesară, dat fiind că decuparea se poate face direct în sistemul normalizat (Fig. 4.17 (c)).



(a) Volumul de decupare în sist. ref. de observare
(b) Volumul de decupare în sist. O'x'y'z'
(c) Volumul de decupare în sist. ref. normalizat

Fig. 4.17 Volumul de decupare în sistemul de vizualizare. (proiecţie în planul x = 0)

În sistemul de referinţă normalizat volumul de decupare este un paralelipiped dreptunghic cu suprafeţele paralele cu planele sistemului de coordonate. Testele de vizibilitate ale unui punct în raport cu planele volumului de decupare sunt:

$$x_N \le 1;\quad x_N \ge -1;\quad y_N \le 1;\quad y_N \ge -1;\quad z_N \le 1;\quad z_N \ge 0; \tag{4.23}$$

Aceste teste se pot efectua direct în coordonatele omogene în sistemul normalizat, fără să mai fie necesară împărţirea cu w_N:

$$\begin{gathered} X_N \le w_N ;\ X_N \ge -\ w_N ; \\ Y_N \le w_N ;\ Y_N \ge -\ w_N ; \\ Z_N \le w_N ;\ Z_N \ge 0; \end{gathered} \tag{4.24}$$

Codul de vizibilitate C al unui vârf în coordonate normalizate omogene se exprimă pe 6 biţi şi fiecare bit se poziţionează în funcţie de rezultatul unuia din testele din relaţia 4.24:

$b_0 = \text{semn}(X_N - w_N)$
$b_1 = \text{semn}(w_N - X_N)$
$b_2 = \text{semn}(Y_N - w_N)$
$b_3 = \text{semn}(w_N - Y_N)$
$b_4 = \text{semn}(Z_N - w_N)$
$b_5 = \text{semn}(Z_N)$

Spaţiul omogen normalizat se împarte în 27 de regiuni după poziţia lor faţă de planele care definesc volumul canonic de vizualizare. În interiorul volumului canonic, toate punctele au codul de vizibilitate cu toţi biţii egali cu 0 (C = 000000). În toate celelalte regiuni, codul de vizibilitate este diferit de 0, având unul sau mai mulţi biţi egali cu 1.

Algoritmul Sutherland-Hodgman de decupare în spaţiu se desfăşoară în mod asemănător cu algoritmul de decupare în plan. Pentru o suprafaţă dată prin vârfurile ei în spaţiu, decuparea relativ la un volumul dat prin ecuaţiile planelor sale se efectuează în modul următor:

(1) Se calculează codurile de vizibilitate ale vârfurilor suprafeţei.
(2) Se efectuează testele de acceptare şi rejecţie sigură. Pentru suprafeţele care nu îndeplinesc nici unul din aceste teste se continuă cu decuparea prin intersecţia cu volumul de decupare.
(3) Decuparea prin intersecţie a unei suprafeţe se efectuează într-un număr de paşi egal cu numărul de plane ale volumului de decupare. În fiecare pas se obţine o nouă suprafaţă, din care a fost eliminată porţiunea invizibilă în raport cu planul de decupare respectiv.

După efectuarea decupării, se calculează coordonatele x_N, y_N, z_N ale vârfurilor suprafeţelor vizibile rezultate (prin împărţirea cu w_N), iar toate celelalte vârfuri (ale suprafeţelor invizibile) sunt ignorate. Acest lucru înseamnă că operaţia de împărţire omogenă se execută pentru un număr strict necesar de vârfuri, ceea ce reprezintă un câştig în viteza de execuţie.

Se pune problema dacă nu ar fi mai simplu să fie efectuată decuparea în plan, deci după calculul proiecţiilor suprafeţelor, dat fiind că algoritmul de decupare în plan este mai simplu. Acest lucru ar însemna ca toate suprafeţele să fie proiectate pe planul de vizualizare, după care să se efectueze decuparea relativ la fereastra de vizualizare. Această soluţie este însă inacceptabilă, datorită faptului că în fereastra se proiecţe se proiectează atât punctele (vârfuri ale suprafeţelor) aflate în volumul de vizualizare, cât şi cele aflate într-un volum simetric cu acesta faţă de planul $z_V = 0$, care se află în spatele observatorului. Tratarea tuturor acestor suprafeţe proiectate în fereastra de vizualizare dar invizibile, consumă un timp de execuţie în mod inutil. De aceea, decuparea obiectelor la volumul de vizualizare nu se poate executa corect şi eficient decât în spaţiul tridimensional, şi anume în coordonate normalizate omogene.

Eliminarea obiectelor aflate în exteriorul volumului de vizualizare

Se poate detalia modul de execuţie a decupării relativ la volumul de vizualizare. Algoritmul de bază se referă la decuparea unei suprafeţe în spaţiu, dată ca o succesiune de vârfuri şi, implicit, o succesiune de segmente de dreaptă orientate (laturile suprafeţei), relativ la un volum de decupare, care este volumul canonic în coordonate omogene normalizate.

Decuparea punctelor în spaţiu este un simplu test de vizibilitate (ecuaţiile 4.24), iar decuparea unei linii este un caz particular (cu n = 2) al algoritmului de decupare a unei suprafeţe.

Decuparea unui obiect tridimensional, modelat ca o colecţie de feţe (suprafeţe plane) se reduce la decuparea fiecărei feţe în parte, iar obiectul rezultat este compus din toate feţele sale vizibile şi părţile rezultate prin decupare.

O creştere a eficienţei în operaţia de decupare se poate obţine folosind teste de rejecţie şi de acceptare sigură nu numai pentru fiecare suprafaţă în parte, dar şi pentru un obiect tridimensional în întregime.

Procedeul de calcul al reuniunii (SC) şi al intersecţiei (PC) codurilor de vizibilitate ale unui obiect tridimensional depinde de modul de reprezentare al acestuia. Pentru obiectele tridimensionale modelate prin feţe date ca listă de indecşi la vârfuri se calculează direct valorile SC şi PC ale celor N vârfuri ale obiectului. Pentru obiecte reprezentate prin feţe separate, se pot colecta codurile de vizibilitate ale feţelor componente şi calcula reunuinea SC şi intersecţia PC. Testul de acceptare sau rejecţie sigură (`Culling`) a unui obiect tridimensional este prezentat în pseudocod asemănător limbajului C astfel:

```
int Culling(){
        PC = 0x3F;
        SC = 0;
        for (i=0; i<N; i++){
                SC = SC | Ci;
                PC = PC & Ci;
        }
        if (PC) {
                Obiect sigur invizibi, va fi rejectat;
                return 0;
        }
        else if (SC){
                Obiect sigur vizibil în întregime;
                Nu se va mai testa fiecare suprafaţă;
                return 1;
        }
        else {
                Se va decupa fiecare suprafaţă;
                return –1;
        }
```

}

Ce anume se execută în fiecare din situaţii depinde, evident, de modul de organizare al programului şi de biblioteca grafică folosită. În orice caz, dacă obiectul este sigur invizibil, atunci se abandonează toate operaţiile referitoare la acest obiect şi se trece la obiectul următor.

Selecţia obiectelor, adică identificarea obiectelor potenţial vizibile pentru redarea imaginii acestora şi eliminarea obiectelor sigur invizibile (*culling*), este o resursă puternică de creştere a eficienţei de redare a scenelor virtuale. În scenele virtuale de dimensiuni mari, care conţin un număr mare de obiecte, în fiecare cadru de imagine sunt vizibile şi deci trebuie să fie prelucrate un număr mult mai mic obiecte decât numărul total de obiecte ale scenei. De aceea, identificarea cât mai devreme posibil a obiectelor sigur invizibile şi eliminarea lor conduce la reducerea substanţială a timpului de redare a scenei. În redarea scenelor complexe, acest procedeu se implementează mai eficient decât modul simplu descris mai sus prin definirea volumului de delimitare a obiectelor.

Volumul de delimitare

Volumul de delimitare (*bounding box*) al unui obiect sau al unui grup de obiecte (ierarhie de obiecte) este un volum definit cât mai simplu, care include toate vârfurile obiectelor şi are dimensiuni minime. Se folosesc ca volume de delimitare paralelipipede dreptunghice sau sfere.

Fiind dat un obiect cu un volum de delimitare, testul de eliminare (*culling*) se efectuează asupra volumului de delimitare: se calculează intersecţia codurilor de vizibilitate ale vârfurilor volumului de delimitare şi, dacă este diferită de zero, atunci volumul de delimitare şi implicit, întregul obiect este invizibil şi abandonat.

Această modalitate de reprezentare şi prelucrare a obiectelor este aproape unanim adoptată în realitatea virtuală, datorită execuţiei eficiente a testului: decizia de eliminare a unui obiect se poate lua prin considerarea unui număr de opt vârfuri, în locul unui număr foarte mare de vârfuri cât are un obiect în mod obişnuit.

Volumul de delimitare al unui obiect se defineşte în sistemul de referinţă local (de modelare) al obiectului, ca un paralelipiped dreptunghic cu muchiile paralele cu axele de coordonate. Un astfel de paralelipiped se poate specifica prin coordonatele a două vârfuri opuse (x_{min}, y_{min}, z_{min}) şi (x_{max}, y_{max}, z_{max}). Vârfurile lui sunt transformate din sistemul de referinţă local în alte sisteme de referinţă (universal, de observare, etc.), la fel ca şi vârfurile obiectului. Calculul codurilor de vizibilitate şi testul de eliminare se poate efectua în sistemul de referinţă de observare sau în alt sistem de referinţă (sistemul normalizat) dacă biblioteca grafică permite accesul la coordonatele în acest sistem.

În Fig. 4.18 sunt reprezentate trei obiecte în poziţii diferite faţă de volumul de vizualizare. Volumul de delimitare al primului obiect este complet exterior faţă de volumul de vizualizare şi obiectul corespunzător ("ceainic" – *teapot*) este ignorat pentru punctul de observare dat. Cel de-al doilea obiect (icosaedru) este complet vizibil, volumul său de delimitare fiind inclus în volumul de vizualizare. Pentru cel de-al treilea obiect (sferă reprezentată wireframe), volumul de delimitare intersectează volumul de vizualizare şi trebuie să fie executată decuparea suprafeţelor sale relativ la volumul de vizualizare.

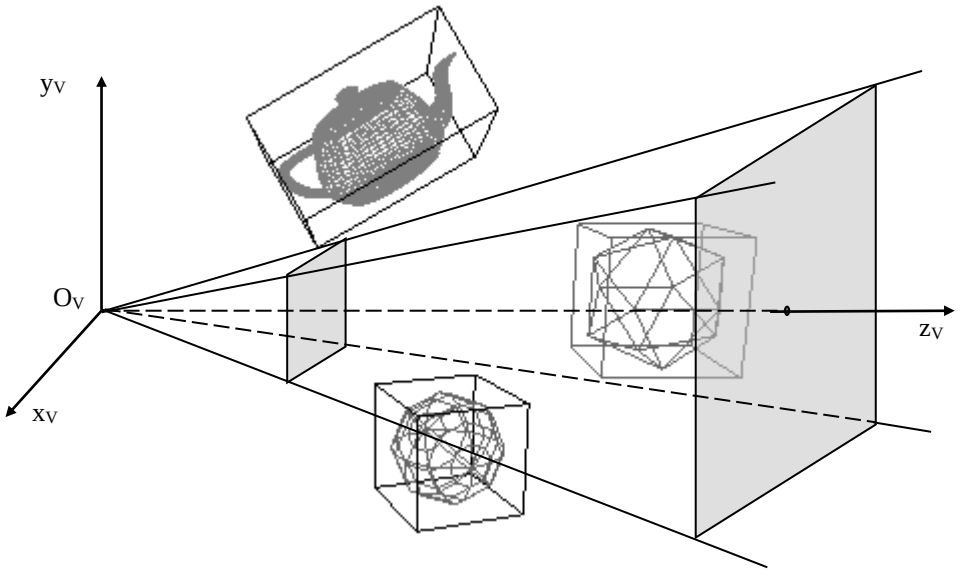


Fig. 4.18 Eliminarea obiectelor (*culling*) pe baza volumului de delimitare (*bounding box*).

În scenele virtuale modelate ierarhic se construiesc volume de delimitare la fiecare nivel de ierarhie şi obiectele sau grupurile de obiecte sunt selectate pe baza testului de eliminare efectuat asupra volumului de delimitare corespunzător.

Detecția coliziunii

Detecţia coliziunii este un aspect important în modelarea mediului virtual, prin faptul că simulează un aspect realist al mediului: obiectele reale nu pot trece unele prin altele; atunci când un obiect se atinge de un altul există un răspuns de coliziune: deformarea obiectelor, schimbarea traiectoriei de deplasare, explozia, etc. La interacţiunea dintre utilizator şi obiectele virtuale se pot produce forţe, vibraţii, şi alte reacţii tactile care sunt prelucrate în sistem.

Detecţia coliziunii se poate calcula aproximativ, folosind volumele de delimitate, sau exact, prin considerarea tuturor punctelor obiectelor. Dat fiind că volumul de delimitare depăşeste în multe puncte suprafaţa de frontieră a obiectului, calculul aproximativ al coliziunii introduce coliziuni false între obiecte: nu obiectele s-au atins ci volumele lor de delimitare.

Generarea exactă a coliziunii se calculează ca a doua etapă după calculul aproximativ al coliziunii. Dacă două obiecte au volumele de delimitare disjuncte, atunci ele sigur nu se ating. Dacă două obiecte au volume de delimitare care se ating sau interpătrund, atunci se calculează coliziunea exactă prin considerarea tuturor punctelor obiectelor

5

Transformarea de rastru

Transformarea de rastru (*rasterization*) este o conversie de la reprezentarea prin coordonatele vârfurilor a unui segment sau a unui poligon (primitivă geometrică), la reprezentarea prin mulţimea corespunzătoare de pixeli care se afişează pe display. Transformarea de rastru se mai numeşte conversie de baleiere (*scan conversion*), sau desenarea segmentelor, respectiv a poligoanelor (*line drawing*, *polygon drawing*) sau generare linie, respectiv poligon (*line generation*, *polygon generation*).

Transformarea de rastru implică trecerea de la spaţiul bidimensional continuu (poarta de afişare definită în planul $x_S = 0$ al sistemului de referinţă ecran 3D) la spaţiul discret al imaginii (Fig. 5.1)

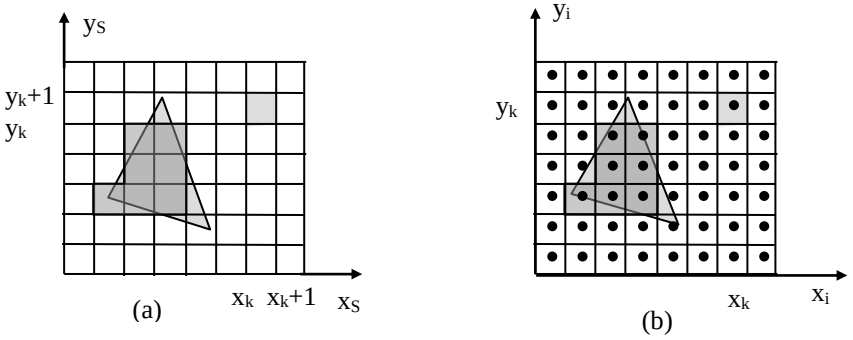


Fig. 5.1 (a) Spaţiul bidimensional continuu al porţii de afişare
(b) Spaţiul bidimensional discret al imaginii

Spaţiul imagine este un tablou bidimensional de locaţii discrete numite pixeli, fiecare pixel având o adresă care corespunde poziţiei în imagine a acesteia. Dimensiunea imaginii este specificată prin numărul de pixeli pe orizontală (M), respectiv pe verticală (N), care pot fi afişaţi în zona de ecran atribuită porţii de afişare. Dimensiunea maximă a imaginii depinde de rezoluţia display-ului folosit.

Un pixel ocupă în poarta de afişare o zonă pătrată de dimensiune 1 x 1 a cărui adresă este dată de coordonata colţului stânga-jos. De exemplu, pixelul de adresă (x_k, y_k) din spaţiul imagine corespunde unui pătrat din poarta de afişare cu colţurile de coordonate (x_k, y_k) (x_k+1, y_k) (x_k+1, y_k+1) (x_k, y_k+1), unde x_k şi y_k sunt numere întregi. În mod simplificat se consideră că imaginea pe ecran a pixelilor este formată din zone pătrate adiacente, dar, în realitate, imaginea pe ecran a unui pixel este o zonă circulară, iar aceste cercuri nu sunt adiacente ci se întrepătrund.

Un pixel din spaţiul imagine se reprezintă prin culoarea lui, care se memorează într-un buffer de imagine. Adresa pixelului în spaţiul imagine determină adresa în bufferul de imagine în care se memorează culoarea acestuia.

Din coordonatele x şi y ale vârfurilor primitivelor geometrice în poarta de afişare se calculează ecuaţiile segmentelor de dreaptă (laturile), iar mulţimea de pixeli generată în spaţiul imagine trebuie să aproximeze cât mai bine segmentele de dreaptă sau interiorul poligoanele delimitat prin laturile sale (Fig.5.1).

Transformarea de rastru a segmentelor sau poligoanelor calculează adresa fiecărui pixel din mulţimea de pixeli prin care se aproximează primitiva geometrică dată şi înscrie la adresa corespunzătoare din bufferul de imagine intensitatea de culoare a pixelului respectiv.

Culoarea pixelului poate fi o culoare constantă, rezultată din atributul de culoare al primitivei geometrice sau din condiţii de umbrire constantă (poligonală), şi atunci această culoare se înscrie la toate adresele pixelilor care aparţin primitivei geometrice în bufferul de imagine. În cazul umbririi Gouraud, culoarea fiecărui pixel se calculează prin interpolare, concomitent cu calculul adresei pixelilor în transformarea de rastru.

Concomitent cu transformarea de rastru, se poate executa operaţia de eliminare a suprafeţelor ascunse, folosind coordonata z_S (adâncimea) vârfurilor primitivelor geometrice calculate în sistemul de referinţă ecran 3D (algoritmul de tip buffer de adâncime (Z-buffer)).

În cele mai multe din sistemele grafice actuale, transformarea de rastru, umbrirea şi eliminarea suprafeţelor ascunse se execută intercorelat, într-o singură scanare (baleiere) a primitivei geometrice. La sistemele evoluate se adaugă la acestea şi operaţiile de netezire a liniilor (anti-aliasing) şi de texturare.

Înainte de descrierea operaţiilor de generare a primitivelor geometrice, se va descrie modul în care sunt reprezentate şi redate culorile în grafica pe calculator.

Reprezentarea culorilor în sistemele grafice

Imaginea care se obţine pe display este compusă dintr-un anumit număr de pixeli, dat de rezoluţia display-ului. Fiecare pixel are o poziţie pe ecran, dată de adresa lui în poarta de afişare, şi o culoare care poate fi reprezentată în mai multe modele: modelul RGB, modelul HSV, modelul HLS, şi altele. Dintre aceste modele, în grafică se foloseşte cel mai frecvent modelul RGB.

Modelul RGB

În modelul RGB, culoarea este reprezentată printr-un triplet de culori primare, roşu (*red*) verde (*green*), albastru (*blue*). Utilizarea preponderentă a modelului RGB în grafică se datorează în primul rând faptului că monitoarele color folosesc acest model de producere a culorii. Culoarea este produsă pe ecranul unui monitor prin excitarea a trei puncte adiacente de substanţă fosforescentă de culori roşu, verde şi, respectiv, albastru. Punctele fosforescente fiind apropiate, ochiul percepe tripletul de puncte ca un singur punct a cărui culoare este suma celor trei componente.

Spaţiul de reprezentare a tuturor culorilor în modelul RGB este un cub într-un sistem de coordonate cu axele notate Red, Green, Blue (Fig. 5.2).

Orice culoare în modelul RGB se exprimă printr-un triplet (r,g,b) şi îi corespunde un punct în spaţiul RGB al cărui vector **C** este:

$$\mathbf{C} = r\mathbf{R} + g\mathbf{G} + b\mathbf{G} \tag{5.1}$$

unde **R**, **G**, **B** sunt versorii axelor Red, Green, Blue. În acest model culoarea negru este reprezentată prin tripletul (0,0,0), iar culoarea alb este reprezentată prin tripletul (1,1,1).

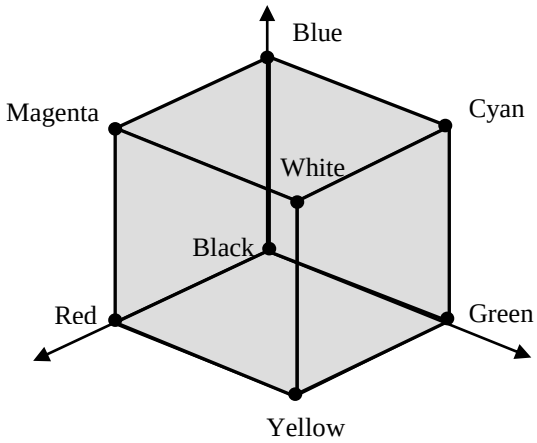


Fig. 5.2 Spaţiul culorilor în modelul RGB

În sistemele grafice se mai foloseşte o variantă a modelului RGB, modelul RGBA, unde cea de-a patra componentă (α) indică transparenţa suprafeţei. Valoarea 1 a acestei componente (α=1) înseamnă suprafaţă opacă, iar valoarea minimă (α=0)

înseamnă suprafaţă complet transparentă. Dacă transparenţa unei suprafeţe este diferită de zero, atunci culoarea care se atribuie pixelilor acestei suprafeţe se modifică în funcţie de culoarea existentă în bufferul de imagine în poziţiile corespunzătoare. Culoarea rezultată a unui pixel este $C' = f(\alpha, C, C_P)$, unde:

- C(r,g,b) este culoarea suprafeţei
- α este transparenţa suprafeţei, cu valori între 0 şi 1
- C_P (r_P, g_P, g_P) este culoarea precedentă a pixelului, aflată în bufferul de imagine
- C'(r',g',b') este noua culoare a pixelului, rezultată prin aplicarea unei funcţii *f* fiecărei componente de culoare

Bibliotecile grafice permit definirea mai multor funcţii *f* de combinare liniară între culoarea transparentă cu culoarea de fond.

Modelul HSV

Modelul HSV reprezintă o transformare neliniară a spaţiului RGB, prin care culorile sunt specificate prin componentele: nuanţă (*hue*), saturaţie (*saturation*) şi valoare (value). Modelul HSV este folosit în definirea mai intuitivă a culorilor (perceptuală), decât selecţia unor culori primare. De exemplu, sarcina de a modifica o culoare astfel încât să fie "mai luminosă", "mai palidă", "mai galbenă", etc, se realizează mult mai simplu în modelul HSV decât în modelul RGB.

Spaţiul culorilor în modelul HSV este o piramidă hexagonală într-un sistem de coordonate polare HVS (Fig. 5.3)

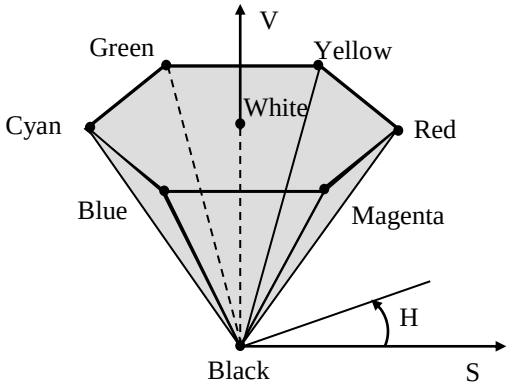


Fig. 5.3 Spaţiul culorilor în modelul HSV.

Definiţiile parametrilor H, S, V sunt legate de modul în care este percepută culoarea:

- Nuanţa (*hue*) este acea calitate prin care ochiul distinge o familie de culori de altă familie de culori, de exmplu roşu de galben, verde de albastru, etc.
- Saturaţia (*saturation*), numită şi cromicitate (*chroma*) este calitatea culorii prin care se distinge o culoare puternică de o culoare slabă. Se mai poate defini ca intensitatea culorii sau distanţa faţă de nuanţa gri.

- Valoare (*value*) este calitatea prin care se distinge o culoare luminoasă de o culoare întunecată.

În modelul HVS, variaţia parametrului H (între 0 şi 360°) corespunde selecţiei culorii. Scăderea parametrului S (desaturarea culorii) înseamnă adăugarea de alb. Scăderea parametrului V (devaluarea culorii) înseamnă adăugare de negru. Conversia din spaţiul RGB în spaţiul HSV se înţelege mai uşor într-o interpretare geometrică: hexagonul din reprezentarea HVS se obţine prin proiectarea cubului RGB pe un plan normal la diagonala principală, pe direcţia diagonalei principale. Corespondenţa dintre vârfurile hexagonului de bază HSV şi vârfurile cubului RGB este:

RGB		HSV
100	red	0,1,1 (H este măsurat în grade)
110	yellow	60,1,1
010	green	120,1,1
011	cyan	180,1,1
001	blue	240,1,1
101	magenta	300,1,1

Pentru aceeşi definiţie a unei culori (de exemplu, prin tripletul rgb) este posibil ca monitoare diferite să genereze culori diferite, datorită caracteristicilor constructive diferite ale monitoarelor. În general, între componenta de culoare transmisă unui monitor şi intensitatea produsă de ecranul monitorului există o relaţie neliniară de forma:

$$R_m = K\left(R_i'\right)^{\chi_r};\ G_m = K\left(G_i'\right)^{\chi_g};\ B_m = K\left(B_i'\right)^{\chi_b} \qquad (5.2)$$

unde χ ia valori între 2.3 şi 2.8.

Linerizarea relaţiei între valorile componentelor culorii produse prin program şi intensităţile generate pe ecran se numeşte corecţie gama (*gamma corection*). Corecţia gama se realizează aplicând o transformare care, compusă cu transformarea 5.3 să asigure o relaţie liniară între culoarea (R_i, G_i, B_i) generată prin program şi culoarea (R_m , G_m, B_m) generată pe ecranul monitorului. Acestă transformare este:

$$R_i' = k\left(R_i\right)^{1/\chi_r};\ G_i' = k\left(G_i\right)^{1/\chi_g};\ B_i' = k\left(B_i\right)^{1/\chi_b} \qquad (5.3)$$

Corecţia gama se poate impelementa printr-o tabelă de căutare (look-up table) încorporată în controlerul grafic, care transformă valorile R_i, G_i, B_i citite din bufferul de imagine în valorile corespunzătoare R_i', G_i', B_i' transmise monitorului. Lipsa corecţiei gama sau o corecţie gama calculată eronat, produce imagini incorecte.

Generarea segmentelor de dreaptă

În orice bibliotecă grafică se găseşte o funcţie care poate fi apelată pentru generarea unui segment de dreaptă. De exemplu funcţia `line()` din biblioteca

compilatorului Borland C, sau funcţia `LineTo()` din interfaţa Win32API, trasează o linie în ferestra de afişare a programului.

Se consideră un segment de dreaptă AB dat prin capetele lui (x_A, y_A), (x_B, y_B) în sistemul de coordonate poarta de afişare (Fig. 5.4). Ecuaţia dreptei care trece prin aceste puncte este $y = ax+b$, unde $a = (y_B-y_A)/(x_B-x_A)$, $b = y_A-ax_A$

În acelaşi sistem de referinţă, adresa unui pixel este dată de coordonatele colţului lui stânga-jos, deci pixelul de adresă (x_k, y_k) este un pătrat cu colţurile (x_k, y_k), (x_k+1, y_k), (x_k+1, y_k+1), (x_k, y_k+1), unde x_k şi y_k sunt numere întregi.

Dacă $x_1 \leq x_2$, atunci se calculează secvenţa de puncte pe dreaptă:

$(x_1 = x_A, y_1 = ax_1+b)$,

$(x_2 = x_1+1, y_2 = ax_2+b)$,....$(x_i, y_i = ax_i+b)$.....

Fiecărui punct pe dreaptă i se asociază un pixel a cărui adresă în sistemul de referinţă imagine se aproximează prin trunchierea sau rotunjirea coordonatelor punctului în sistemul de referinţă de afişare:

$x_i' = (int)x_i$ sau $x_i' = (int)(x_i+0.5)$

$y_i' = (int)y_i$ sau $y_i' = (int)(y_i+0.5)$

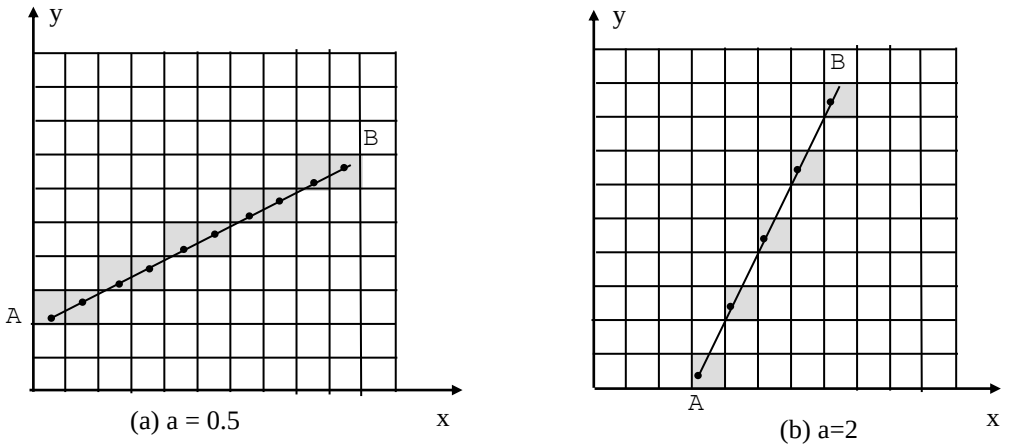


Fig. 5.4 Generarea segmentelor de dreaptă prin aplicarea directă a relaţiilor de calcul.

În acest mod se obţine transformarea în spaţiul imagine a unui segment de dreaptă, reprezentat prin mulţimea pixelilor care aproximează segmentul de dreaptă dat.

Aplicarea directă a formulei de calcul a punctelor de pe dreaptă are dezavantajul că necesită mai multe operaţii (adunare, înmulţire) în numere flotante pentru fiecare pixel, operaţii care au timp de execuţie ridicat. Un alt dezavantaj al acestei metode este că generează segmente de dreaptă întrerupte dacă a > 1 (Fig. 5. 4(b)).

Înlăturarea acestor inconveniente se face prin utilizarea algoritmilor incrementali în care se evită repetarea operaţiilor de înmulţire în virgulă flotantă, folosind trecerea prin incrementare de la un punct la următorul. Pentru obţinerea

segmentelor de dreaptă fără întrerupri se tratează separat cazurile $abs(a) \leq 1$ şi $abs(a)>1$.

Dacă $abs(a) \leq 1$ şi $x_A \leq x_B$, se începe generarea din punctul $x_1 = x_A$, $y_1 = y_A$ şi la fiecare pas se incrementeză x cu 1, deci se iau punctele de pe dreaptă care au: $x_1=x_A$, $x_2=x_1+1$, $\ldots x_i, x_{i+1}=x_i+1, \ldots..$ Dat fiind că $y_i = ax_i+b$, rezultă că $y_{i+1} = ax_{i+1}+b = a(x_i+1)+b$, deci $y_{i+1} = y_i+a$. Deci, pentru trecerea de la un punct la următorul, x se incrementează cu 1, iar y se incrementează cu valoarea pantei. Punctului de pe dreaptă de coordonate (x_i, y_i), îi corespunde un pixel a cărui adresă în spaţiul imagine se calculează prin trunchiere sau rotunjire.

Dacă $abs(a) \leq 1$ şi $x_A > x_B$, se începe generarea din punctul $x_1 = x_B$, $y_1 = y_B$ şi restul calculelor se continuă la fel ca în cazul precedent.

Dacă $abs(a) >1$ şi $x_A \leq x_B$, se începe generarea segmentului din punctul $x_1 = x_A$, $y_1 = y_A$ şi la fiecare pas se incrementează y cu 1, deci se calculează punctele care au: $y_1=y_A$, $y_2=y_1+1, \ldots. y_i$, $y_{i+1} = y_i+1, \ldots$ Deoarece $x_i=(y_i-b)/a$, rezultă că $x_{i+1} =(y_{i+1}-b)a = x_i+1/a$. Deci, pentru trecerea de la un punct la următorul, y se incrementează cu 1, iar x se incrementează cu inversul pantei dreptei.

Dacă $abs(a) > 1$ şi $x_A > x_B$, se începe generarea din punctul $x_1 = x_B$, $y_1 = y_B$, şi restul calculelor se continuă la fel ca în cazul precedent.

Algoritmul incremental de generare a primitivelor geometrice are mai multe variante de implementare, în funcţie de situaţia în care este aplicat. Pentru segmente de dreaptă, se adaptează algoritmul incremental astfel încât să se genereze segmente fără întreruperi, folosind numai operaţii cu numere întregi. Această variantă este cunoscută sub numele de algoritmul Bresenham. Pentru generarea poligoanelor, varianta de algoritm incremental este algoritmul de baleiere pe linii (*scan-line*).

Algoritmul Bresenham se bazează pe metoda incrementală, dar este optimizat astfel încât foloseşte numai operaţii cu numere întregi. Datorită acestei proprietăţi, el poate fi implementat hardware şi, în majoritatea cazurilor, operaţiile de generare a segmentelor de dreaptă din acceleratoarele grafice se bazează pe o implementare hardware a algoritmului Bresenham. Descrierea detaliată a algoritmului Bresenham se poate găsi în bibliografie [Mold96].

Indiferent de algoritmul de conversie folosit pentru generarea liniilor, trecerea de la spaţiul bidimensional continuu, în care poziţia punctelor este reprezentată prin numere reale, la reprezentarea în spaţiul discret al imaginii, este o aproximare prin eşantionare, care produce erori de reprezentare. Din cauza acestei aproximări, orice linie dreaptă (cu excepţia celor orizontale sau verticale) apare zimţată (*jagged*), ca o "scară" de segmente de dreaptă succesive, aşa cum se poate observa în orice imagine generată pe un display sau reprodusă pe imprimantă, inclusiv în textul de faţă. Imaginea de scară este cu atât mai pregnantă cu cât rezoluţia display-ului (numărul de pixeli pe orizontală şi pe verticală) este mai mică. Dacă rezoluţia creşte, aspectul nedorit de scară este atenuat, dar nu este eliminat. Netezirea liniilor se poate obţine prin diferite metode de filtrare, metode cunoscute sub numele de filtrare anti-aliasing.

Generarea poligoanelor

Metoda cea mai frecvent folosită pentru generarea mulţimii pixelilor prin care se aproximează o suprafaţă poligonală este metoda de baleiere pe linii a poligonului (algoritmul *scan-line*). Principiul de bază al acestei metode pentru un poligon convex este prezentat în Fig. 5.5.

Se consideră poligonul $A'B'C'D'E'$, care reprezintă proiecţia în planul $z_S=0$ al sistemului de referinţă ecran 3D, a unei suprafeţe $ABCDE$ din spaţiul tridimensional universal. Poligonul este baleiat cu o linie orizontală care ia valori succesive, y_1, $y_2, \ldots y_i$, y_{i+1}.., unde y_1 este valoarea coordonatei y minimă a vârfurilor poligonului, iar $y_{i+1} = y_i+1$. Pentru fiecare poziţie y_i a liniei de baleiere, se calculează intersecţiile $x_{i,1}$, $x_{i,2}$ cu două din laturile poligonului. Pixelii a căror adrese sunt cuprinse în intervalul $(x_{i,1}, y_i)$, $(x_{i,2}, y_i)$, aparţin poligonului şi sunt prelucraţi în mod corespunzător. Pentru algoritmul simplu de generare a poligoanelor, culoarea poligonului (cunoscută ca atribut a feţei corespunzătoare a obiectului) este înscrisă în bufferul de imagine la toate adresele cuprinse în intervalul respectiv. În algoritmii în care se combină generarea cu eliminarea suprafeţelor ascunse sau umbrire, se prelucrează toţi pixelii corespunzători acestui interval.

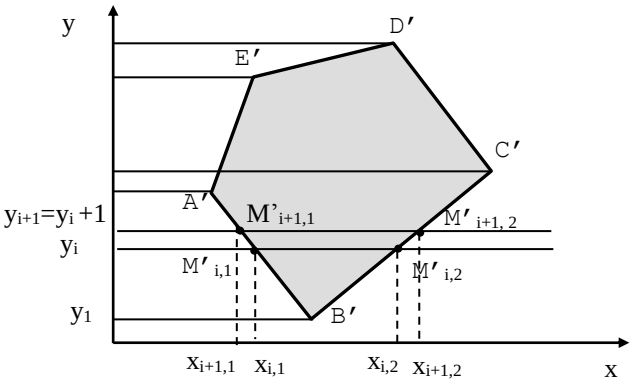


Fig. 5.5 Principiul baleierii pe linii a poligoanelor (scan-line)

Fie următoarele ecuaţii ale laturilor stânga şi dreapta cu care se intersectează dreapta $y = y_i$:

$$x = m_1 y + n_1$$
$$x = m_2 y + n_2 \qquad (5.4)$$

Pentru situaţia din Fig.5.5, intersecţia dreptei $y = y_i$ are loc cu laturile $A'B'$ şi $B'C'$, pentru care coeficienţii din ecuaţiile de mai sus au valorile:

$$m_1 = (x_A - x_B)/(y_A - y_B), \quad n_1 = x_B - m_1 y_B$$
$$m_2 = (x_C - x_B)/(y_C - y_B), \quad n_2 = x_C - m_2 y_C \qquad (5.5)$$

Intersecţiile corespunzătoare sunt: $x_{i,1} = m_1 y_i + n_1$, $x_{i,2} = m_2 y_i + n_2$. Coordonatele intersecţiilor liniei următoare de baleiere $y = y_{i+1}$, unde

$y_{i+1} = y_i + 1$, se pot calcula din coordonatele precedente folosind coeficienţii m_1 şi m_2 ai laturilor intersectate:

$$x_{i+1,1} = x_{i,1} + m_1$$
$$x_{i+1,2} = x_{i,2} + m_2 \qquad (5.6)$$

Secvenţa de pixeli ai căror adrese sunt cuprinse în intervalul $(x_{i+1,1}, y_{i+1})$, $(x_{i+1,2}, y_{i+1})$ aparţin poligonului pe linia de baleiere $y = y_{i+1}$ şi sunt trataţi corespunzător.

Acesta este principiul de bază al baleierii pe linii (scan-line). Cazurile particulare, când se schimbă latura cu care are loc intersecţia (la colţurile poligonului), sau când există laturi orizontale, se tratează separat.

Algoritmul de baleiere pe linie se poate implementa şi pentru poligoane concave. În acest caz, o linie de baleiere poate interesecta mai multe perechi de laturi, rezultând mai multe segmente de interecţie şi deci mai multe secvenţe de pixeli care aparţin poligonului (Fig. 5.6(a)). În general, însă, poligoanele concave sau cu găuri se tratează pe mai multe nivele. Acceleratoarele grafice implementează hardware automate de generare a unor suprafeţe convexe simple (triunghiuri sau trapeze). Descompunerea unui poligon convex în mai multe triunghiuri este o operaţie simplă care, de cele mai multe ori, este preluată de biblioteca grafică, care pune la dispoziţie funcţii de generare a poligoanelor convexe cu un număr oarecare de laturi. Descompunerea poligoanelor concave în poligoane convexe este tratată la nivelul programului de aplicaţie. Această ierarhizare în prelucrarea poligoanelor este precizată în secţiunea următoare pentru biblioteca OpenGL.

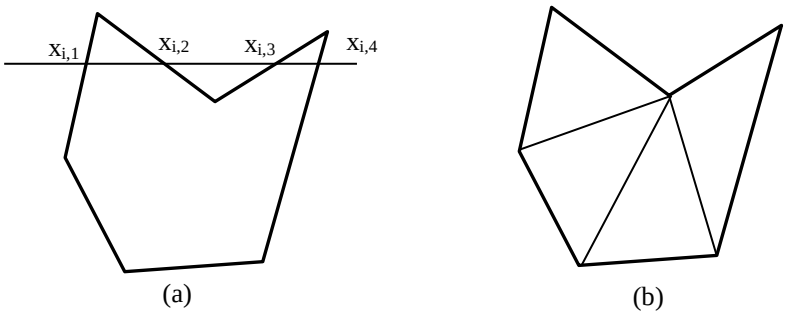


Fig. 5.6 Generarea poligoanelor concave:
(a) generare directă prin baleiere pe linii;
(b) transformare în mulţime de triunghiuri (triangularizare).

Ca şi în cazul liniilor, muchiile poligoanelor generate prezintă aspectul de scară de segmente, datorită trecerii la spaţiul discret al imaginii. Algoritmii de filtrare pentru netezirea muchiilor poligonului (algoritmi de anti-aliasing) se execută, în general, concomitent cu algoritmul de generare scan-line a poligonului.

Eliminarea suprafeţelor ascunse

Reprezentarea pe display a tuturor suprafeţelor obiectelor proiectate în fereastra de vizualizare produce o imagine nerealistă deoarece este posibil ca o suprafaţă care se află în spatele altei suprafeţe să fie vizibilă

pe ecran. Acestă situaţie provine din transformarea de proiecţie din spaţiul tridimensional în spaţiul bidimensional (planul de vizualizare). În general, orice proiecţie înseamnă o reducere a dimensionalităţii spaţiului de reprezentare şi, prin acesta, se pierd o parte din proprietăţile formelor. Datorită proiecţiei, suprafeţe distincte, aflate la distanţe diferite în spaţiu, sunt proiectate în aceeaşi regiune pe display şi sunt reprezentate suprapus. Pentru obţinerea unor imagini corecte, care să reflecte relaţia spaţială dintre obiecte, se prelucrează suplimentar suprafeţele proiectate ale obiectelor.

Prelucrarea suprafeţelor din punct de vedere al poziţiei relative între ele, prelucrare cunoscută sub numele de eliminarea suprafeţelor ascunse (*hidden surface removal*) se poate efectua în două moduri:

(a) Eliminarea suprafeţelor ascunse în spaţiul obiect, prin compararea adâncimii relative a suprafeţelor obiectelor în sistemul de referinţă de observare sau în sistemul de referinţă normalizat.

(b) Eliminarea suprafeţelor ascunse în spaţiul imagine, prin compararea adâncimii pixelilor imaginii rezultaţi din generarea (conversia de rastru) a tuturor suprafeţelor obiectelor, prin algoritmul cunoscut sub numele de algoritm de tip buffer de adâncime (*depth buffer*, *Z-buffer*).

În momentul actual, eliminarea suprafeţelor ascunse prin algoritmul Z-buffer este aproape universal folosită în grafică, datorită calităţilor remarcabile ale acestei abordări, care permite implementări hardware eficiente şi este independentă de modul de reprezentare a obiectelor scenei. Alţi algoritmi de eliminare a suprafeţelor ascunse se mai folosesc doar în aplicaţii grafice foarte specializate, care nu beneficiază de mecanismul de Z-buffer.

Compararea adâncimilor

Pentru eliminarea suprafeţelor ascunse se compară adâncimea (distanţa faţă de punctul de observare) a tuturor suprafeţelor sau a pixelilor rezultaţi prin conversia de rastru.

Un punct P este ascuns de o suprafaţă S, dacă adâncimea punctului este mai mare decât adâncimea intersecţiei I a dreaptei care uneşte punctul P cu centrul de proiecţie (se consideră centrul de proiecţie identic cu originea sistemului de referinţă de observare O_V , Fig. 5.7(a)), deci dacă $O_VP > O_VI$.

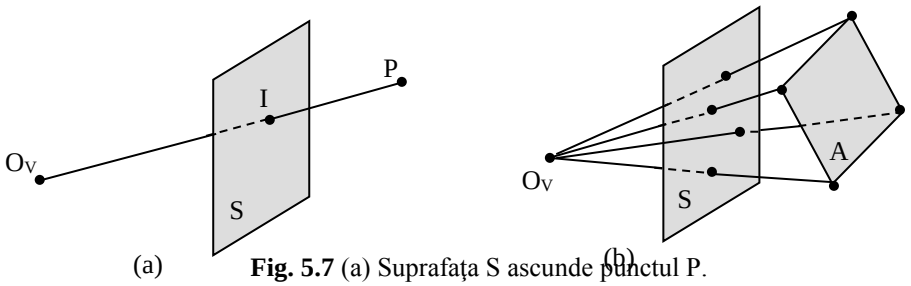


Fig. 5.7 (a) Suprafaţa S ascunde punctul P.
(b) Suprafaţa A ascunde suprafaţa B.

O suprafaţă A este ascunsă de altă suprafaţă S dacă toate punctele suprafeţei A sunt ascunse de suprafaţa S (Fig. 5.7(b)). Dacă o suprafaţă ascunde S suprafaţa A, atunci se consideră că adâncimea suprafeţei S este mai mică decât cea a suprafeţei ascunse A.

Compararea adâncimilor a două puncte în sistemul de referinţă de observare este destul de complicată, deoarece comparaţia are sens numai dacă cele două puncte se află pe aceeaşi linie de proiecţie (Fig. 5.7). Această operaţie este mai simplă în sistemul de referinţă ecran 3D. În acest sistem, $z_S = z_N$, iar z_N este definit de relaţia 4.10 astfel:

$$z_S = z_N = \frac{f}{f - d}\left(1 - \frac{d}{z_V}\right)$$

Acestă relaţie transformă, de exemplu, un cub în sistemul de referinţă de observare, într-un trunchi de piramidă în sistemul ecran 3D (Fig. 5.8).

Proiecţia perspectivă din sistemul de referinţă de observare se transformă în proiecţie paralelă în sistemul de referinţă ecran 3D. Deoarece $z_V = 0$ se transformă în $z_S = -\infty$, liniile de proiecţie în sistemul ecran 3D

sunt linii paralele îndreptate către $-\infty$. Adâncimea unui punct $P_1(x_1,y_1,z_1)$ în sistemul de referinţă de observare, care este dată de distanţa $O_V P_1 = \sqrt{x_1^2 + y_1^2 + z_1^2}$, se transformă în coordonata z_S a punctului în sistemul de referinţă ecran 3D.

Testul de adâncime din sistemul de referinţă de observare, pentru a determina dacă un punct P_1 este mai aproape de centrul de proiecţie (care este, în general, acelaşi cu punctul de observare) decât punctul P_2, necesită ca cele două puncte să se afle pe aceeaşi linie de proiecţie. În sistemul ecran 3D, toate punctele care au aceleaşi coordonate (x_S, y_S) în planul de proiecţie se află pe acelaşi proiector, o dreaptă paralelă cu axa z_S, deci se poate efectua testul de adâncime prin comparaţia coordonatelor z_S ale punctelor.

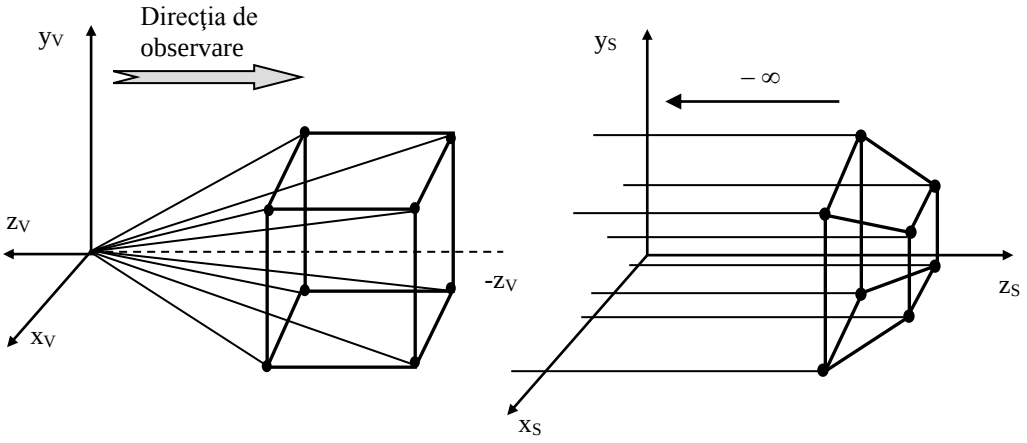


Fig. 5.8 Transformarea unui cub şi a liniilor de proiecţie din sistemul de referinţă de observare în sistemul de referinţă ecran 3D.

Se poate trage concluzia că operaţiile care implică testarea sau compararea adâncimilor se execută mai simplu în sistemul de referinţă 3D. Acest lucru este utilizat atât în algoritmii de eliminare a suprafeţelor ascunse în spaţiul obiect cât şi în algoritmii în spaţiul imagine.

Eliminarea suprafeţelor ascunse în spaţiul obiect

Unul din cei mai cunoscuţi algoritmi de eliminare a suprafeţelor ascunse în spaţiul obiect este algoritmul propus de Newell, cunoscut şi sub numele de algoritmul pictorului, datorită analogiei cu modul în care un pictor creează un tablou. În acest algoritm, se sortează suprafeţele în ordinea descrescătoare a adâncimii şi se redau în această ordine pe display. Eliminarea suprafeţelor ascunse se obţine prin suprapunerea noii suprafeţe (mai apropiată) peste suprafeţele mai depărtate, deja desenate. Lista ordonată de suprafeţe (numită listă de display) se creează în sistemul de referinţă de observare sau în sistemul de referinţă ecran 3D, după operaţia de decupare.

Ordonarea suprafeţelor după adâncime este o operaţie complicată, mai ales dacă două suprafeţe se intersectează între ele, situaţie în care nu se poate decide care suprafaţă este mai apropiată, decât prin divizarea uneia din ele de-a lungul liniei de intersecţie.

Un alt algoritm de eliminare a suprafeţelor ascunse în spaţiul obiect este algoritmul cu subdivizarea recursivă a poligoanelor (algoritmul Weiler-Atherton). Spre deosebire de algoritmul pictorului, în care intersectarea suprafeţelor proiectate se efectuează prin supraînscrierea suprafeţelor mai apropiate peste cele mai depărtate, în algoritmul Weiler-Atherton se execută o operaţie de decupare în sistemul de referinţă normalizat între toate suprafeţele obiectelor şi se reţin numai suprafeţele "învinătoare", adică cele mai apropiate. Lista acestora (lista de display) nu mai conţine suprafeţe care se pot suprapune pe ecran şi nu mai trebuie să fie ordonată. Ca şi în algoritmul pictorului, operaţiile de creare a listei de display necesită un timp de execuţie foarte mare, care depinde de numărul de suprafeţe care se prelucrează. Acest lucru limitează drastic posibilitatea de utilizare a algoritmilor în spaţiul obiect.

Eliminarea suprafețelor ascunse în spațiul imagine: Algoritmul Z-buffer

Dezvoltat de Catmull (1975), algoritmul Z-buffer introduce o metodă simplă de eliminare a suprafeţelor ascunse, în care se efectuează comparaţia între adâncimea z_S a fiecărui pixel (x, y) al poligoanelor generate şi se selectează pixelul cu adâncimea minimă (cel mai apropiat).

Această selecţie se implementează cel mai convenabil prin folosirea unui buffer de adâncime (Z-buffer) care memorează adâncimea tuturor pixelilor generaţi în spaţiul de imagine, a căror culoare este memorată în bufferul de imagine. În orice moment, la fiecare adresă (x, y) în Z-buffer, este memorată adâncimea celui mai apropiat pixel de adresă (x, y) rezultat în transformarea de rastru a poligoanelor. Adâncimea unui nou pixel generat este comparată cu adâncimea pixelului cu aceeaşi adresă din Z-buffer şi noul pixel înlocuieşte pixelul anterior numai dacă adâncimea lui este mai mică. Înlocuirea pixelului anterior cu un pixel nou generat se realizează prin înscrierea culorii noului pixel în bufferul de imagine la adresa (x, y) şi înscrierea adâncimii lui la aceeaşi adresă (x, y) în Z-buffer.

Comparaţia între adâncimile punctelor în sistemul de referinţă ecran 3D are avantajul că poate fi efectuată prin comparaţia coordonatelor z_S, dat fiind că toate punctele cu aceleaşi coordonate (x, y) se află pe aceeaşi linie de proiecţie paralelă cu axa z_S.(Fig. 5.8). Dar această simplitate de calcul are, totuşi, un inconvenient, şi anume acela că adâncimile din sistemul de referinţă de observare nu se transformă liniar în sistemul de referinţă ecran 3D. Intervale egale în z_V nu se transformă în intervale egale în z_S, ceea ce se poate observa şi din relaţiile de transformare 4.10. Cu cât z_V se apropie de planul de vizibilitate depărtat (far), z_S se apropie mai rapid de 1. Obiectelele sunt comprimate în funcţie de coordonata lor z_V, şi acest lucru are consecinţe asupra preciziei în comparaţia adâncimilor.

Algoritmul Z-buffer se execută concomitent cu transformarea de rastru a poligoanelor. Fiecare poligon este descris prin lista vârfurilor sale reprezentate în sistemul de referinţă ecran 3D. Adâncimea pixelilor poligonului se calculează prin interpolare pornind de la adâncimea vârfurilor acestuia.

Interpolarea pentru calculul adâncimii pixelilor se efectuează în două etape: o interpolare de-a lungul laturilor poligonului, prin care se obţine adâncimea pixelilor de pe conturul poligonului, şi o interpolare de-a lungul fiecărei liniei de baleiere, prin care se obţine adâncimea tuturor pixelilor interiori. Algoritmul de generare a poligoanelor prin baleiere pe linii (scan-line) se transformă astfel într-un algoritm combinat de baleiere şi Z-buffer (*scan-line Z-buffer*).

Se reia algoritmul de generare a poligoanelor prin baleiere pe linii din paragraful 5.3 şi Fig.5.5. Prin proiecţia suprafeţei ABCDE din spaţiu în planul $x_S = 0$ se obţine poligonul A''B''C''D''E'' (Fig. 5.9). Liniile de baleiere ale acestui poligon sunt dreptele de ecuaţii $y = y_1$, $y = y_2$,….$y = y_i$, $y = y_{i+1}$, …Dreapta $y = y_i$ interesctează laturile A''B'' şi B''C'' în punctele $M''_{i,1}$ $(y_i, z_{i,1})$ şi $M''_{i,2}$ $(y_i, z_{i,2})$.

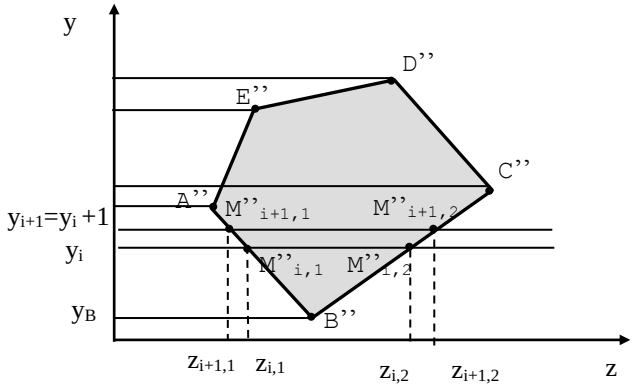


Fig. 5.9 Baleierea poligonului în planul yz.

Ecuaţiile laturilor A''B'' şi B''C'' se pot scrie similar cu relaţiile 5.4:

$$z = m_{1z}y+n_{1z}$$
$$z = m_{2z}y+n_{2z} \qquad (5.7)$$

unde: $m_{1z} = (z_A-z_B)/(y_A-y_B)$, $n_{1z} = y_B-m_{1z}\,z_B$

$$m_{2z} = (z_C-z_B)/(y_C-y_B),\ n_{2z} = y_C-m_{1z}\,z_C \qquad (5.8)$$

Intersecţiile dreptelor A''B'' şi B''C'' cu dreapta $y = y_i$ au coordonatele $z_{i,1} = m_{1z}y_i+n_{1z}$ şi $z_{i,2} = m_{2z}y_i+n_{2z}$. Coordonatele punctelor de intersecţie ale dreptei $y = y_{i+1} = y_i+1$, se pot calcula din coordonatele intersecţiilor precedente folosind coeficienţii m_{1z} şi m_{2z} ai laturilor intersectate:

$$z_{i+1,1} = z_{i,1} + m_{1z}$$
$$z_{i+1,2} = z_{i,2} + m_{2z} \qquad (5.9)$$

Pentru fiecare linie de baleiere se calculează coordonatele intersecţiilor cu laturile poligonului prin incrementarea valorilor precedente cu coeficienţii daţi de relaţiile (5.8).

Cele două operaţii de baleiere, pentru calculul adresei şi al adâncimii pixelilor de pe conturul unei suprafeţei ABCDE, se execută concomitent prin intersecţia suprafeţei cu planele de ecuaţii $y = y_1$, $y = y_2$, ... y_i, y_{i+1}, ... Laturile AB şi BC sunt intersectate de planul $y = y_i$ în punctele $M_{i,1}$ şi $M_{i,2}$. Intersecţiile $(x_{i,1}, y_i)$, $(x_{i,2}, y_i)$ se calculează în planul $z_S = 0$, prin intersecţia dreptei $y = y_i$ cu proiecţiile A'B' şi B'C' ale laturilor AB şi BC în planul $z_S = 0$. Intersecţiile $(z_{i,1}, y_i)$, $(z_{i,2}, y_i)$ se calculează în planul $x_S = 0$, prin intersecţia dreptei $y = y_i$ cu proiecţiile A''B'' şi B''C'' ale laturilor AB şi BC în planul $x_S = 0$.

Adresele pixelilor se obţin prin rotunjirea (sau trunchierea) coordonatelor punctelor de intersecţie $(x_{i,1}, y_i)$, $(x_{i,2}, y_i)$; valorile $z_{i,1}$ şi $z_{i,2}$ reprezintă adâncimea acestor pixeli, care se află pe conturul poligonului.

Pentru calculul adâncimii pixelilor interiori (a celor ce nu se află pe conturul poligonului), se efectuează o interpolare pe fiecare linie de baleiere.

În planul $y = y_i$, relaţia dintre coordonatele x şi z ale dreptei $M_{i,1}M_{i,2}$ este:

$$z = m_{ix}x+n_{ix} \qquad (5.10)$$

unde: $m_{ix} = (x_{i,2} - x_{i,1})/(z_{i,2} - z_{i,1})$, $n_{ix} = z_{i,1} - m_{ix}\,x_{i,1}$

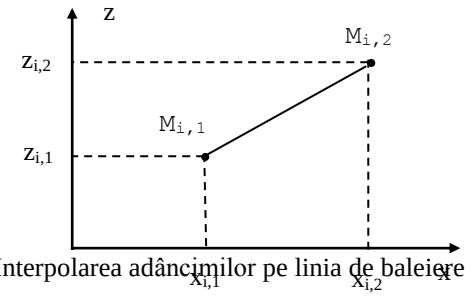


Fig. 5.10 Interpolarea adâncimilor pe linia de baleiere $y = y_i$.

Un punct pe această dreaptă are coordonatele $x_{i,j}$, $z_{i,j} = x_{i,j}\, m_{ix} + n_{ix}$. Pe segmentul de dreaptă $M_{i,1}M_{i,2}$ se calculează puncte succesive prin incrementarea cu 1 a coordonatei x şi calculul prin incrementare cu panta m_{ix} a coordonatei z:

$$x_{i,j+1} = x_{i,j} + 1;\ z_{i,j+1} = z_{i,j} + m_{ix} \qquad (5.11)$$

Valoarea $z_{i,j}$ reprezintă adâncimea pixelului a cărui adresă se obţine prin rotunjirea sau trunchierea coordonatelor punctului ($x_{i,j}$, y_i).

Toate aceste operaţii se execută pentru fiecare faţă a obiectelor tridimensionale în sistemul de coordonate ecran 3D, şi pentru fiecare pixel se actualizează în mod corespunzător culoarea în bufferul de imagine şi adâncimea în bufferul de adâncime. Exprimate în pseudocod asemănător limbajului C, operaţiile combinate de conversie de baleiere şi Z-buffer (funcţia Scanline) arată astfel:

```
void Scanline(){
        for (toate valorile (x,y)){
                image_buffer[x][y] = culoare_stergere;
                Z_buffer[x][y] = max_z; /* initializare Z-
buffer */
        }
        for (fiecare poligon){
                calculul ymin, ymax ale poligonului;
                for (yi = ymin; yi<= ymax;yi++){
                        calcul xi,1, xi,2
                        calcul zi,1, zi,2
                        for (x=xi,1;x<xi,2;x++){
                calcul z;
                if(z < Z_buffer[x][y]){
                                Z_buffer[x][y] = z;
                   image_buffer[x][y] = culoare;
                                }
                        }
                }
        }
                                }
```

Cel mai important avantaj al algoritmului Z-buffer este simplitatea implementării lui, iar dezavantajul principal îl constitue memoria suplimentară necesară pentru implementarea bufferului de adâncime. Dimensiunea fiecărei locaţii (fiecare locaţie corespunzând unui pixel din imagine) este dată de precizia de reprezentare a adâncimii z. Datorită compresiei pe care o introduce transformarea de la sistemul de referinţă de observare la sistemele de referinţă normalizat şi ecran 3D (relaţia 4.10), este posibil ca două valori z_V distincte să fie transformate în valori z_S identice. În mod obişnuit, se folosesc 20 sau 32 de biţi pentru reprezentarea adâncimii unui pixel, ceea ce înseamnă că Z-bufferul poate deveni mai mare decât bufferul de imagine, construit în mod obişnuit cu 24 de biţi/pixel.

În realizările mai vechi ale sistemelor grafice, bufferul de adâncime era implementat în memoria principală. În momentul actual, sunt disponibile terminale grafice care au Z-bufferul realizat ca o memorie dedicată, şi aceasta reprezintă cea mai bună soluţie.

Un alt avantaj al Z-bufferului îl reprezintă independenţa de forma de reprezentare a obiectelor. Descrierea dată mai sus se referă la utilizarea Z-bufferului în redarea obiectelor modelate prin suprafeţe poligonale, dar aceasta nu este o restricţie, deoarece se poate calcula adâncimea z pentru orice tip de suprafaţă, iar mecanismul de comparaţie a adâncimilor este acelaşi.

Eliminarea suprafeţelor orientate invers

Algoritmii de eliminare a suprafeţelor ascunse prezentaţi mai sus se ocupă de compararea suprafeţelor provenind de la oricâte obiecte diferite, pentru care trebuie să fie calculate adâncimile şi sortate (în spaţiul obiect) sau comparate (în spaţiul imagine) pentru a decide care suprafaţă ascunde o altă suprafaţă.

Există însă şi o situaţie de ascundere între suprafeţe care se poate rezolva mult mai simplu, şi anume ascunderea reciprocă între feţele unui obiect opac.

În modelarea obiectelor tridimensionale, feţele componente sunt reprezentate orientat în spaţiu, astfel încât normalele tuturor feţelor unui obiect să fie îndreptate către aceeaşi regiune a spaţiului (spre interiorul sau spre exteriorul obiectului).

Dacă un obiect opac ale cărui normale la feţe sunt îndreptate spre exterior, este privit dintr-un punct de observare, feţele ale căror normale sunt orientate către punctul de observare ascund (maschează) feţele ale căror normale sunt orientate în direcţie inversă.

Orientarea consistentă a feţelor unui obiect permite selecţia numai a acelor feţe ale obiectului care sunt orientate către punctul de observare. Această operaţie de selecţie (numită eliminarea feţelor orientate invers - "din spate" - *backface elimination*) se efectuează în sistemul de referinţă de observare (Fig. 5.11). Pentru testul de vizibilitate al feţelor în funcţie de orientare se calculează produsul scalar dintre normala la suprafaţă şi un vector dintr-un punct al feţei spre punctul de observare. Dacă normalele la feţele obiectului sunt îndreptate către exterior (sens pozitiv, invers acelor de ceas) şi unghiul dintre cei doi vectori este mai mic de 90°, atunci suprafaţa este vizibilă.

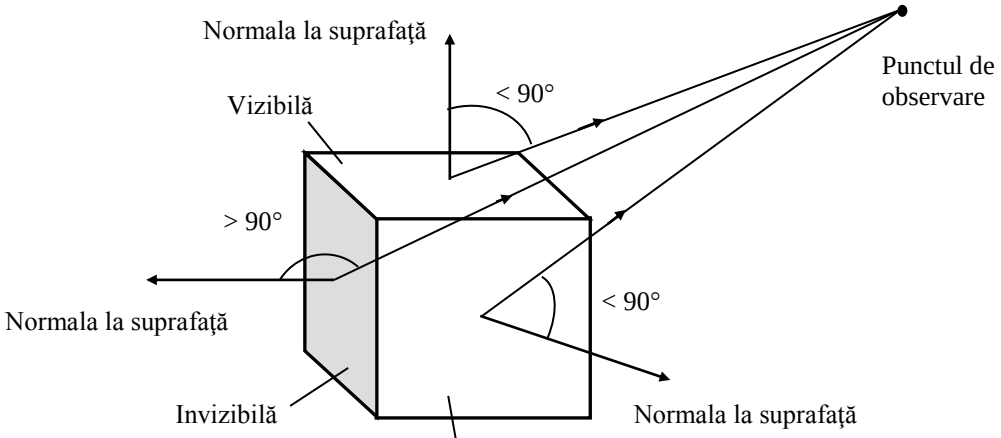


Fig. 5.11 Selecţia feţelor în funcţie de orientare

Produsul scalar a doi vectori, **N** şi **V** este:

$$\mathbf{V} \bullet \mathbf{N} = |\mathbf{V}||\mathbf{N}| \cos\theta$$

unde |**V**| şi |**N**| sunt modulele celor doi vectori, iar θ este unghiul dintre ei. Se poate scrie deci:

V•N > 0 dacă θ < 90°
V•N = 0 dacă θ = 90°

$V \bullet N < 0$ dacă $\theta > 90°$

De aici rezultă condiţia de vizibilitate a unei feţe: pentru ca o faţă să fie vizibilă, produsul scalar dintre normala acesteia şi un vector dintr-un punct al feţei spre punctul de observare trebuie să fie pozitiv:

$$\mathbf{N} \bullet \mathbf{V} > 0 \qquad (5.12)$$

Dacă se consideră normala la suprafaţă orientată către interiorul obiectului (în sens negativ, în direcţia acelor de ceas), atunci condiţia de vizibilitate a unei feţei este ca produsul scalar dintre normală şi un vector dintr-un punct al feţei spre punctul de observare trebuie să fie negativ.

Eliminarea feţelor orientate invers se execută în sistemul de referinţă de observare; orice faţă care este invizibilă este ignorată pentru operaţiile ulterioare de redare. Acest test este foarte important deoarece, în medie, jumătate din feţele unui obiect tridimensional sunt orientate invers faţă de punctul de observare şi pot fi eliminate din calcul.

Dacă obiectul are feţe transparente, atunci nu se pot elimina feţele în funcţie de orientare şi toate feţele obiectului trebuie să fie luate în considerare şi prelucrate corespunzător. Pentru o faţă dată prin succesiunea vârfurilor sale $V_1, V_2, \ldots V_n$, normala se calculează ca produs vectorial a vectorilor care reprezintă două laturi succesive (Fig. 5.12 (a)).

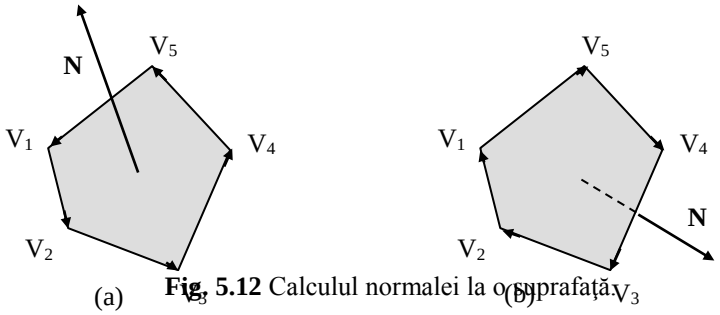


Fig. 5.12 Calculul normalei la o suprafaţă.

$$\mathbf{N} = (V_2 - V_1) \times (V_3 - V_2)$$

Dacă (x_m, y_m, z_m) reprezintă coordonatele vârfului V_m , , iar **i**, **j**, **k** sunt versorii axelor x,y,z atunci normala **N** are expresia:

$$\mathbf{N} = (V_2 - V_1) \times (V_3 - V_2) = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_2 & y_3 - y_2 & z_3 - z_2 \end{vmatrix}$$

Se dezvoltă determinantul după prima linie, şi rezultă:

$$\begin{aligned} \mathbf{N} = {} & ((y_2 - y_1)(z_3 - z_2) - (y_3 - y_2)(z_2 - z_1))\,\mathbf{i} + \\ & ((x_3 - x_2)(z_2 - z_1) - (x_2 - x_1)(z_3 - z_2))\,\mathbf{j} + \\ & ((x_2 - x_1)(y_3 - y_2) - (x_3 - x_2)(y_2 - y_1))\,\mathbf{k} \end{aligned} \qquad (5.13)$$

Dacă ordinea de parcurgere a vârfurilor feţei se inversează, deci se consideră faţa $V_n, \ldots V_2 V_1$, atunci normala se calculează prin produsul $\mathbf{N}=(V_2-V_3)\times(V_1-V_2)$ care are semnul inversat (Fig. 5.12 (b)), ceea ce se regăseşte în formula de calcul a produsului vectorial.

Rezultă că sensul vectorului normal la o faţă este dat de ordinea de parcurgere a vârfurilor feţei. Dacă în sistemul de referinţă local (de modelare) feţele unui obiect sunt definite (prin ordinea vârfurilor) astfel încât toate normalele să fie îndreptate spre aceeaşi regiune a spaţiului (spre exteriorul obiectului), această orientare se păstrează după transformarea de instanţiere şi observare şi în sistemul de referinţă de observare obiectele vor avea normalele orientate consistent (toate orientate către exterior). Această condiţie este suficientă pentru calculul vizibilităţii fiecărei feţe prin produs scalar al normalei cu vectorul de direcţie de la un punct al feţei către punctul de observare.

Eliminarea suprafeţelor orientate invers ale obiectelor opace trebuie să fie efectuată în orice sistem de vizualizare, deoarece este un test care se execută eficient şi elimină în medie jumătate din suprafeţele obiectelor din scenă.

Dacă scena este compusă dintr-un singur obiect convex opac, eliminarea feţelor orientate invers este suficientă pentru a rezolva problema ascunderii feţelor invizibile. Dacă, însă, obiectul este concav, este posibil ca o faţă orientată direct (către punctul de observare) să fie parţial mascată de o faţă orientată invers şi atunci eliminarea suprafeţelor orientate invers nu mai este suficientă, fiind necesar unul din algoritmii de eliminare a suprafeţelor ascunse care se bazează pe calculul adâncimii. La fel, dacă în scenă există mai mult de un obiect, pe lângă eliminarea suprafeţelor orientate invers, este necesară eliminarea suprafeţelor ascunse.

Anti-aliasing

Se poate observa uşor că liniile afişate pe display apar ca o secvenţă de puncte care dau un aspect zimţat (în trepte de scară), aspect cunoscut sub numele de *aliasing*. În text se va păstra termenul original de aliasing, dat fiind că traducerile sunt mai puţin sugestive.

Cauza fundamentală a aliasing-ului în grafica pe calculator este crearea imaginilor prin eşantionare în domeniul spaţiului şi, pentru imaginile animate, şi în domeniul timp. Pentru generarea unei imagini, pasul final îl constitue calculul intensităţii pixelilor, care reprezintă trecerea de la intensitatea `I(x,y)` definită în spaţiul bidimensional continuu, la intensitatea fiecărui pixel, în spaţiul imagine discret. Această trecere este echivalentă cu eşantionarea spaţiului bidimensional continuu cu o grilă de puncte de eşantionare, considerate, de exemplu, în centrul fiecărui pixel

Cea mai familiară manifestare a aliasing-ului îl constitue aspectul de asperitate (*jagged*) al liniilor sau al muchiilor poligoanelor. De asemenea, suprafeţele înguste pot apare întrerupte, în funcţie de orientarea lor. Un alt efect al aliasing-ului îl constitue apariţia şi dispariţia obiectelor mici, care ocupă o suprafaţă mai mică decât suprafaţa unui pixel şi care sunt redate sau nu în funcţie de intersecţia cu grila de eşantionare.

Aceste efecte sunt mai supărătoare în secvenţele de imagini în mişcare. Muchiile în formă de scară sau suprafeţele întrerupte dau o senzaţie de mişcare necontrolată a acestora (*crowl*), care este intolerabilă în multe aplicaţii de realitate virtuală, de exemplu în simulatoarele de zbor.

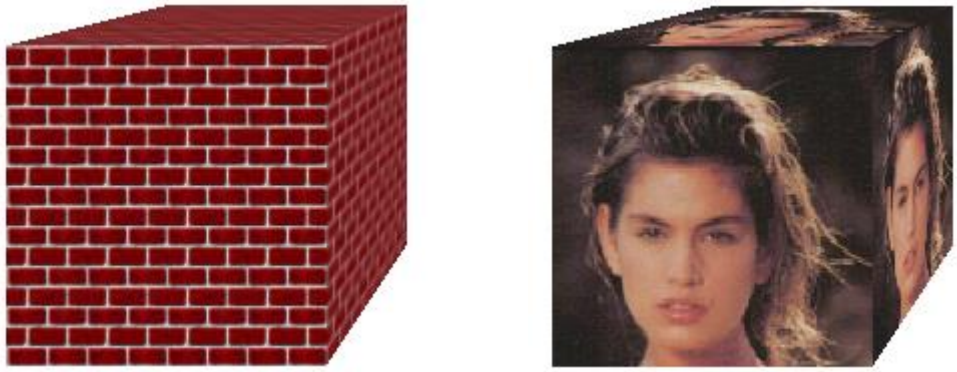
Tehnicile de prelucrare a imaginii pentru atenuarea efectelor de aliasing sunt cunoscute sub numele de anti-aliasing. Toţi algoritmii folosiţi pentru diminuarea efectului de aliasing necesită resurse de memorie şi de calcul considerabile şi, cu cât se admit aproximări mai grosiere, cu atât scade calitatea imaginii datorită zgomotului de spectru transpus. În secţiunea 7 este prezentată o modalitate de netezire a muchiilor poligoanelor implementată în biblioteca grafică OpenGL.

Texturarea

Aplicarea texturilor este o tehnică importantă de creştere a realismului scenelor virtuale, prin care se generează imagini ale obiectelor tridimensionale mult

mai interesante şi mult mai complexe. Prin texturare se modulează culoarea suprafeţelor obiectelor folosind o imagine repetitivă (textura), obţinându-se un efect asemănător aceleia de aplicare (lipire) a unui desen pe o suprafaţă.

În decursul dezvoltării tehnicilor de texturare, au fost propuse şi alte modalităţi de creare a impresiei de texturare, de exemplu prin perturbaţia normalelor suprafeţelor, sau prin variaţii ale iluminării mediului ambiant al obiectelor. În momentul actual, metoda cea mai larg râspîndită, cu numeroase implementări hardware în sistemele grafice, este prin modularea culorii suprafeţelor obiectelor folosind imagini de texturare. Texturile care modulează culoarea suprafeţelor măresc informaţia vizuală prezentată prin sugerarea materialului suprafeţei. De exemplu, prin aplicarea repetată a imaginii unei cărămizi (obţinută, de cele mai multe ori prin scanarea unei fotografii a unui perete real), un singur dreptunghi capătă aspectul unui perete întreg (Fig. 5.13(a)). Desenarea unui astfel de perete folosind câte un poligon pentru fiecare cărămidă este extrem de ineficientă şi imaginea obţinută este nerealistă, deoarece fiecare cărămidă este prea netedă şi regulată. De asemenea, prin texturare se poate aplica pe o suprafaţă o imagine care este practic imposibil să fie generată prin suprafeţe poligonale (Fig. 5.13(b)).



(a) (b)

Fig. 5.13 Suprafeţe texturate prin aplicarea unei texturi bidimensionale

Pentru texturarea obiectelor prin modularea culorii suprafeţelor trebuie să se definească următoarele caracteristici:

- Modul de aplicare a texturii pe suprafaţa unui obiect.
- Modul de diminuare (eliminare) a efectelor de aliasing care apar în aplicarea texturilor (filtrarea texturilor).

Aplicaţia texturilor

Aplicaţia texturii (*texture mapping*) asigură redarea corectă a poligoanelor, atunci când li se atribuie o textură. De exemplu, atunci când un perete texturat este observat în perspectivă, cărămizile mai depărtate se văd mai mici decât cele apropiate.

Texturile pot fi definite ca imagini bidimensionale, care se aplică suprafeţelor obiectelor, sau ca un câmp tridimensional, şi atunci culoarea obiectului se determină prin intersecţia suprafeţelor sale cu acest câmp tridimensional al texturii. Texturile tridimensionale sunt mai rar implementate în sistemele grafice, de aceea în continuare vor fi prezentate numai texturile bidimensionale.

Textura bidimensională este o imagine definită într-un spaţiu bidimensional (numit spaţiul texturii) T(s, t), în intervalul [0,1] pe ambele axe de coordonate s şi t. Textura este alcătuită din elementele componente discrete care sunt dreptunghiuri (sau pătrate) numite texeli. Fiecare imagine de textură este definită prin rezoluţia ei, dată ca număr de texeli pe cele două coordonate. De exemplu, se folosesc imagini de textură de 256 x 256 texeli, 512 x 1024, 1024 x 1024, etc.

Aplicaţia texturii bidimensionale unui obiect constă din parametrizarea suprafeţelor, urmată de transformarea obiectului în sistemul de referinţă ecran 3D (Fig. 5.14). Prin parametrizarea suprafeţelor se asociază un punct de coordonate (s, t) în spaţiul texturii fiecărui vârf al obiectului modelat (reprezentat în sistemul de referinţă de modelare). Transformarea completă de aplicaţie a texturii mai necesită, după parametrizarea suprafeţelor, toate transformările geometrice obişnuite, de la sistemul de referinţă model (spaţiul obiect) la sistemul de referinţă ecran 3D.

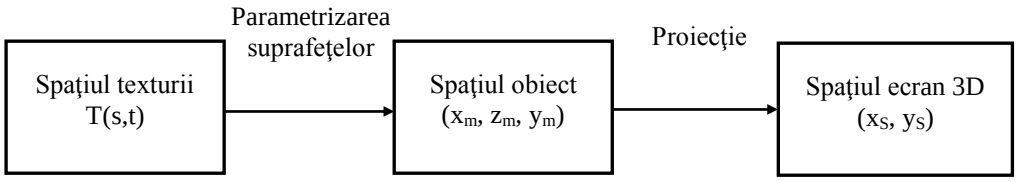


Fig. 5.14 Operaţiile de aplicaţie a texturilor bidimensionale.

Dacă se compun cele două transformări (parametrizarea suprafeţei şi transformarea în spaţiul ecran 3D) se obţine o transformare de aplicaţie a texturii (*texture mapping*), care transformă puncte din spaţiul texturii în spaţiul ecran 3D. În Fig. 5.15 (a), textura este o tablă de şah, definită în spaţiul texturii. Această textură este aplicată unor suprafeţe prin definirea coordonatelor fiecărui vârf al suprafeţei în planul texturii, iar prin transformarea în spaţiul ecran 3D şi transformarea de rastru, se obţine imaginea suprafeţelor texturate (Fig. 5.15 (b1, b2, c)).

Transformarea de parametrizare amplasează imaginea texturii pe un obiect prin asocierea unui punct (s, t) în spaţiul texturii (parametrul de texturare), fiecărui punct (x_m, y_m, z_m) al obiectului. Modul cum poate fi colată o suprafaţă plană (textura) pe un obiect tridimensional nu are o soluţie unică şi nu poate fi realizat decât admiţând diferite tipuri de erori şi distorsiuni. În implementările practice au fost adoptate mai multe tehnici specifice aplicaţiilor. De exemplu, se poate atribui câte o textură fiecărei feţe a unui obiect (aşa cum este în Fig. 5.13), unui grup de feţe ale obiectului, sau se poate proiecta aceeaşi textură pe toate feţele obiectului.

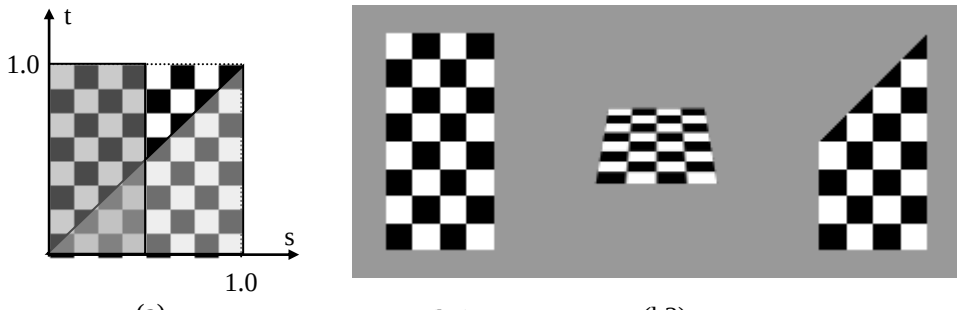


Fig. 5.15 Transformarea de aplicaţie a texturilor.
(a) Spaţiul texturii. (b1), (b2), (c) Suprafeţe texturate în spaţiul ecran 3D.

Pentru aplicaţia texturii pe o suprafaţă plană, se definesc coordonatele (s, t) în spaţiul texturii pentru fiecare vârf al suprafeţei. Aceste coordonate sunt calculate o singură dată, la modelarea obiectului, şi sunt memorate împreună cu coordonatele vârfului şi cu normala în vârf. Dacă $0 \leq s, t \leq 1$, atunci imaginea texturii apare (în întregime sau parţial) pe suprafaţa texturată. De exemplu, în Fig. 5.15, prima suprafaţă texturată, are coodonatele vârfurilor în spaţiul texturii (0.0, 0.0), (0.5, 0.0) (0.5, 1.0), (0.0, 1.0), şi imaginile din Fig 5.15(b1) şi (b2) se obţin prin două transformări de modelare diferite (amplasare în spaţiul universal). Cea de a doua suprafaţă are coordonatele vârfurilor în spaţiul texturii (0.5, 1.0), (1.0, 0.0), (1.0, 1.0), (0.5, 0.5) şi imaginea din Fig. 5.15(c) pentru un anumit punct de observare.

Dacă coordonatele vârfurilor suprafeţei în spaţiul texturii depăşesc pătratul cu latură egală cu 1 în care este definită imaginea de textură, atunci se poate ca textura să se repete pe suprafaţa texturată. De exemplu, în imaginea din Fig. 5.13(a), faţa verticală a cubului are coordonatele de textură ale vârfurilor (0.0, 0.0), (4.0, 0.0), (4.0, 4.0), (0.0, 4.0). Textura (formată din două "cărămizi") se repetă de patru ori pe suprafaţă (se văd opt cărămizi).

Adresa texelilor corespunzători pixelilor unei primitive geometrice (obţinuţi prin transformarea de rastru), se calculează printr-o *transformare inversă transformării de aplicaţie a texturii.* Un pixel, definit ca un pătrat în spaţiul ecran 3D, este transformat în spaţiul model obiect printr-o transformare inversă transformării model-ecran 3D, şi apoi este transformat în spaţiul texturii, printr-o transformare inversă transformării de parametrizare. Deoarece transformarea inversă transformării model-ecran 3D nu este lineară, unui pixel din imagine îi corespunde o suprafaţă în spaţiul texturii care, în general, este un patrulater. Culoarea texelului corespunzător acestui patrulater este atribuită pixelului.

Calculul texelilor în spaţiul texturii corespunzători pixelilor primitivelor geometrice este un proces de eşantionare şi este posibil să apară un zgomot de spectru transpus (aliasiang), atunci când frecvenţa de eşantionare este mai mică decât dublul frecvenţei spaţiale maxime a texturii. De aceea trebuie să fie folosite tehnici de anti-aliasing, adică filtrarea texturilor. Aceste aspecte sunt prezentate în paragraful următor.

Filtrarea texturilor

Fenomenul de aliasing care apare în texturare are cauza în eşantionarea imaginii de textură. Transformarea din spaţiul ecran 3D în spaţiul texturii bidimensionale generează o suprafaţă în formă de patrulater corespunzătoare fiecărui pixel, care este numită pre-imaginea pixelului. Pentru simplificarea prezentării se presupune că pre-imaginea pixelului este un pătrat în spaţiul texturii (Fig. 5.16).

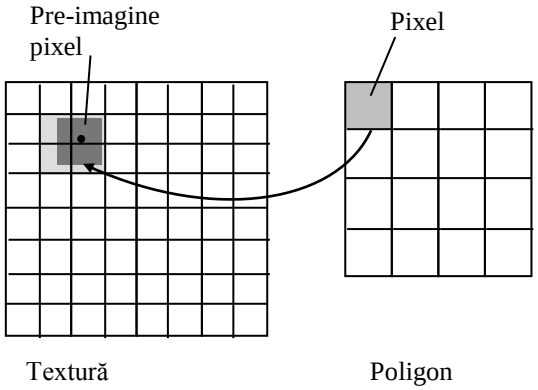


Fig. 5.16 Pre-imaginea unui pixel în spaţiul texturii.

Dacă nu se efectuează filtrarea texturii, atunci se atribuie pixelului culoarea texelului cel mai apropiat de centrul pre-imaginii acestuia. Probelemele de aliasing care apar în această situaţie sunt foarte grave, mai ales pentru imaginile suprafeţelor privite în perspectivă, în care variază mult dimensiunea pre-imaginii pixelilor.

Pre-imaginea unui pixel în planul texturii depinde de poziţia suprafeţei faţă de punctul şi direcţia de observare. Se presupune o suprafaţă pătrată cu parametrii texturii (0,0), (1,0), (1,1), (0,1), perpendiculară pe direcţia de observare, la o distanţă D astfel că imaginea ei pe ecran este un pătrat de N x N pixeli. Se mai presupune, de asemenea, că textura folosită are rezoluţia de N x N texeli. În această situaţie, aria pre-imaginii unui pixel în spaţiul texturii este egală un aria unui texel (Fig. 5.17).

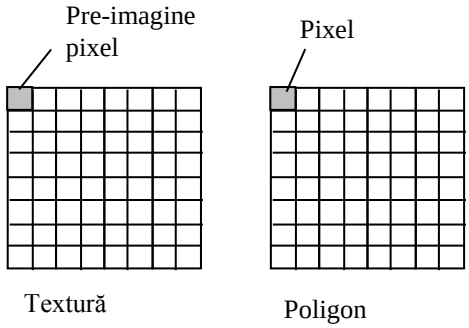


Fig. 5.17 Pre-imaginea unui pixel cu arie egală cu aria texelului

Dacă suprafaţa se apropie de punctul de observare, la distanţă D/2, atunci imaginea suprafeţei pe ecran este de 2N x 2N pixeli şi imaginea celor 2N x 2N pixeli

acoperă N x N texeli, deci aria imaginii unui pixel este acum ¼ din aria unui texel (Fig. 5.18). Cu cât suprafaţa se apropie mai mult, cu atât scade dimensiunea pre-imaginii pixelului în planul texturii şi, deci creşte frecvenţa de eşantionare (inversul dimensiunii pre-imaginii pixelului) a texturii. În această situaţie nu apare zgomot de aliasing, dar imaginea nu devine mai bună, chiar dacă se efectuează filtrarea prin medierea ponderată a culorilor texelilor acoperiţi de pre-imaginea pixelului. Această situaţie se datorează faptului că textura nu este o imagine continuă, ci este reprezentată prin elemente discrete (texelii) şi se observă discontinuităţile de trecere de la un texel la altul.

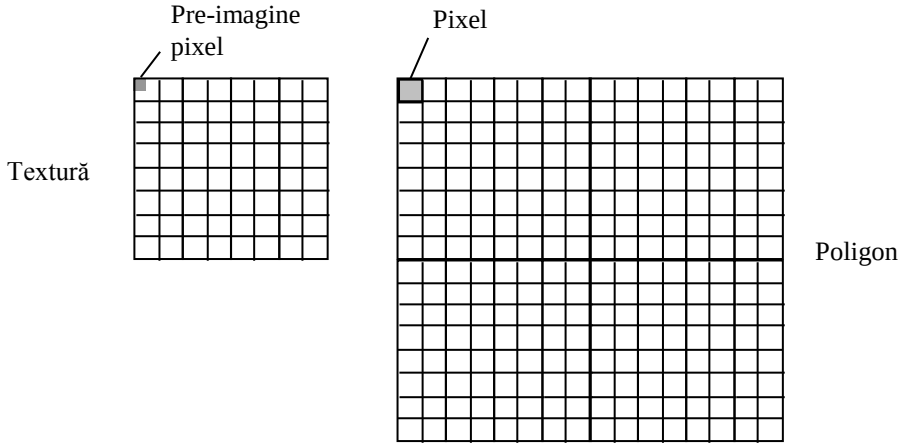


Fig. 5.18 Pre-imaginea unui pixel cu arie mai mică decât aria texelului

Dacă suprafaţa se depărtează de punctul de observare, de exemplu la distanţa 2D, atunci suprafaţa are dimensiunea de (N/2) x (N/2) pixeli pe ecran este şi imaginea acestora acoperă N x N texeli, deci aria pre-imaginii unui pixel este de patru ori mai mare decât aria unui texel (Fig. 5.19).

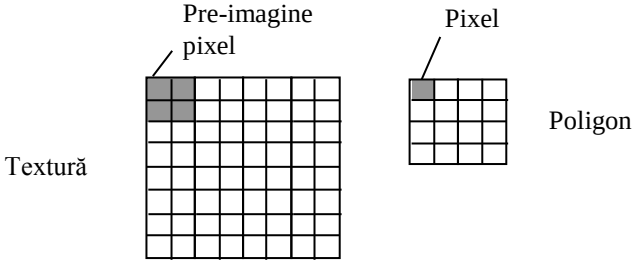


Fig. 5.19 Pre-imaginea unui pixel cu aria mai mare decât aria unui texel.

Cu cât suprafaţa se depărtează mai mult, cu atât creşte dimensiunea pre-imaginii pixelului în planul texturii şi, deci, scade frecvenţa de eşantionare a texturii (inversul dimensiunii pre-imaginii pixelului). În această situaţie apare zgomot de aliasing, dacă nu se efectuează filtarea imaginii texturii. Filtrarea se execută într-o fereastră Fourier pe dimensiunea pre-imaginii unui pixel în planul texturii, şi se efectuează prin medierea ponderată a intensităţilor tuturor texelilor acoperiţi de pre-imaginea pixelului.

Prefiltrarea texturilor. Procedeul de filtrare a texturii poate fi accelerat datorită faptului că imaginea care se filtrează (textura) este cunoscută şi se pot construi imagini prefiltrate ale acesteia. În loc să se efectueze medierea pe suprafaţa mai multor texeli atunci când pre-imaginea pixelului acoperă un nunăr mare de texeli, se poate folosi o imagine de textură prefiltrată, cu rezoluţie mai scăzută, în care culoarea fiecărui nou texel reprezintă media culorilor texelilor din textura originală.

O tehnică elegantă de prefiltrare a texturilor, care are o mare popularitate şi numeroase implementări hardware în acceleratoarele grafice, este tehnica numită *mip-map*, dezvoltată de Wiliams în 1983. În loc să fie folosită o singură imagine de textură, în tehnica mip-map se foloseşte o succesiune de imagini ale aceleiaşi texturi, toate derivate din imaginea originală, prin medierea culorilor şi reducerea rezoluţiei. Fiecare imagine din succesiune are o rezoluţie egală cu jumătatera rezoluţiei imaginii precedente şi fiecare texel este media a patru texeli din imaginea precedentă.. Termenul *mip* provine din expresia *multum in parvo* (multe lucruri într-un spaţiu restrâns). Parametrul L este parametrul de selecţie a nivelului de filtrare a texturii. Nivelul L = 0 reprezintă imaginea originală a texturii (imaginea nefiltrată) cu rezoluţia maximă. În general, în tehnica mip-map se folosesc rezoluţii egale pe cele două coordonate s şi t, cu valoare o putere a lui 2, iar imaginile prefiltrate au rezoluţii care scad prin înjumătăţire până la valoarea 1 x 1 (Fig. 5.20).

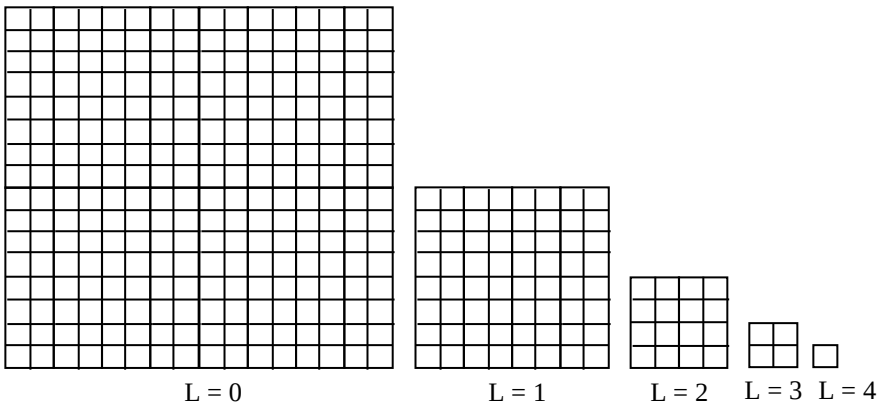


Fig. 5.20 Textură mip-map

Prin selecţia nivelului de filtrare a texturii (L), se alege imaginea cu rezoluţia cea mai potrivită pentru dimensiunea pre-imaginii pixelului. Pentru evitarea discontinuităţilor între imagini la rezoluţii diferite, se pot combina două nivele de filtrare prin interpolare liniară. Selecţia corectă a nivelului L este importantă. Dacă L este prea mare, imaginea apare estompată; dacă L este prea mic, atunci se observă zgomotul de aliasing. Nivelul L se alege astfel ca aria pre-imaginii pixelului să fie cât mai apropiată de aria texelului.

Detaliile de execuţie ale prefiltrării folosind texturi mip-map depind de implementarea specifică, hardware sau software. În secţiunea următoare sunt prezentate funcţiile bibliotecii OpenGL prin care se pot programa aplicaţii grafice folosind texturarea suprafeţelor.

6

Biblioteca grafică OpenGL

Pentru programarea aplicaţiilor grafice complexe necesare în crearea şi redarea scenelor virtuale, se pot utiliza în momentul de faţă mai multe biblioteci şi interfeţe grafice, precum şi sisteme de dezvoltare de programe (toolkit-uri), care permit proiectantului să reutilizeze un număr mare de funcţii grafice deja implementate şi să-şi concentreze eforturile asupra aplicaţiei însăşi. Dat fiind că majoritatea acestor biblioteci sunt foarte iefine sau accesibile gratis prin Internet şi pot fi folosite într-un număr mare de platforme hardware şi software, cunoaşterea şi utilizarea lor este deosebit de importantă şi utilă.

Dintre bibliotecile grafice existente, biblioteca OpenGL, scrisă în limbajul C, este una dintre cele mai utilizate, datorită faptului că implementează un număr mare de funcţii grafice de bază pentru crearea aplicţiilor grafice interactive, asigurând o interfaţă independentă de platforma hardware.

Introdusă în anul 1992, biblioteca OpenGL a devenit în momentul de faţă cea mai intens folosită interfaţă grafică pentru o mare varietate de platforme hardware şi pentru o mare varietate de aplicaţii grafice 2D şi 3D, de la proiectare asistată de calculator (CAD), la animaţie, imagistică medicală şi realitate virtuală. Datorită suportului oferit prin specificaţiile unui consorţiu specializat (OpenGL Architecture Review Board) şi a numărului mare de implementări existente, în momentul de faţă OpenGL este în mod real singurul standard grafic multiplatformă. Calităţile de maturitate, stabilitate, portabilitate şi fiabilitate se adaugă la cele de compatibilitate între versiunile existente şi de uşurinţă de utilizare datorită unei bune structurări şi a documentaţiei care se găseşte cu uşurinţă. Au fost publicate numeroase cărţi, exemple de cod şi alte informaţii ieftine sau accesibile liber pe reţeaua internet. Una din adresele internet care se pot folosi pentru astfel de informaţii este `www.opengl.org`.

Aplicaţiile OpenGL pot rula pe platforme foarte variate, începând cu PC, staţii de lucru şi până la supecalculatoare, sub cele mai cunoscute sisteme de operare: Linux, Unix, Windows NT, Windows 95, Mac OS. De asemenea se integrează în orice sistem de administrare a ferestrelor (windowing), incluzând Win32, X/Windows, şi Presentation Manager. Funţiile OpenGL sunt apelabile din limbajele Ada, C, C++ şi Java. Ca rezultat, aplicaţiile scrise pentru OpenGL pot fi adaptate pentru orice sistem disponibil al utilizatorilor. Din consorţiul de arhitectură OpenGL fac parte reprezentanţi de la firmele Compaq, Evens-Sutherland, Hewlett-Packard, IBM, Intel, Intergraph, Microsoft şi Silcon Graphics. Pentru scrierea aplicaţiilor folosind interfaţa OpenGL de către utilizatori finali nu este necesară obţinerea unei licenţe.

Implementările bibliotecii OpenGL diferă de la un sistem grafic la altul, dezvoltatorii sistemelor având astfel posibilitatea de optimiza costul şi performaţele acestora. Funcţiile OpenGL, care sunt aceleaşi ca interfaţă de apel, pot fi executate de hardware specializat, pot fi executate ca rutine soft de procesorul unităţii centrale CPU, sau pot fi implementate ca o combinaţie de rutine software şi hardare specializat. Această flexibilitate de implementare a permis realizarea unei game variate de acceleratoarele grafice OpenGL, de la simple plăci de redare grafică, până la implemetarea completă a pipeline-ului grafic, inclusiv, funcţiile de Z-buffer, antialising, ceaţă, texturare. Există acceleratoare grafice OpenGL de la plăci cu preţ scăzut pentru calculatoare PC, până la sisteme grafice ale staţiilor de lucru puternice şi multiprocesoare.

În lucrarea de faţă sunt prezentate aspectele cele mai importante ale programării aplicaţiilor grafice folosind biblioteca OpenGL, cu intenţia de a asigura un minim de cunoştinţe de pornire a experienţei de programare. De asemenea, sunt descrise programele folosite pentru exemplificarea aspectelor teoretice de modelare şi generare a imaginilor (prezentate în secţiunile 2, 3, 4 şi 5). Prezentarea completă a tuturor funcţiilor bibliotecii OpenGL nu este nici posibilă, nici necesară. Numai ghidul de programare OpenGL (*OpenGL Programming Guide*, [Woo97]) conţine peste 600 de pagini, la care se adaugă manualele de referinţă şi multe altele. Dar, odată înţelese principiile de organizare a interfeţei, restul detaliilor de implementare se pot parcurge cu uşurinţă din orice material disponibil sau help on-line.

Folosind biblioteca OpenGL, un programator în domeniul graficii se poate concentra asupra aplicaţiei dorite şi nu mai trebuie să fie preocupat de detaliile de implementare a diferitelor funcţii “standard”. Programatorul nu este obligat să scrie algoritmul de generare a liniilor sau a suprafeţelor, nu trebuie să calculeze decuparea obiectelor la volumul de vizualizare, etc, toate acestea fiind deja implementate în funcţiile bibliotecii şi, cel mai probabil, mult mai eficient şi adaptat platformei hardware decât le-ar putea realiza el însuşi.

În redarea obiectelor tridimensionale, biblioteca OpenGL foloseşte un număr redus de primitive geometrice (puncte, linii, poligoane), iar modele complexe ale obiectelor şi scenelor tridimensionale se pot dezvolta particularizat pentru fiecare aplicaţie, pe baza acestor primitive.

Dat fiind că OpenGL prevede un set puternic, dar de nivel scăzut, de comenzi de redare a obiectelor, mai sunt folosite şi alte biblioteci de nivel mai înalt care

utilizează aceste comenzi şi preiau o parte din sarcinile de programare grafică. Astfel de biblioteci sunt:

- Biblioteca de funcţii utilitare GLU (OpenGL Utility Library) permite definirea sistemelor de vizualizare, redarea suprafeţelor curbe şi alte funcţii grafice.
- Pentru fiecare sistem Windows există o extensie a bibliotecii OpenGL care asigură interfaţa cu sistemul respectiv: pentru Microsoft Windows, extensia WGL, pentru sisteme care folosesc sistemul X Window, extensia GLX, pentru sisteme IMB OS/2, extensia PGL.
- Biblioteca de dezvoltare GLUT (OpenGL Utility Toolkit) este un sistem de dezvoltare independent de sistemul Windows, care ascunde dificultăţile interfeţelor de aplicaţii Windows, punând la dispoziţie funcţii pentru crearea şi iniţializarea ferestrelor, funcţii pentru prelucrarea evenimentelor de intrare şi pentru execuţia programelor grafice bazate pe biblioteca OpenGL.

În orice aplicaţie OpenGL trebuie să fie incluse fişierele header `gl.h` şi `glu.h` sau `glut.h` (dacă este utilizată biblioteca GLUT). Toate funcţiile bibliotecii OpenGL încep cu prefixul `gl`, funcţiile `GLU` încep cu prefixul `glu`, iar funcţiile GLUT încep cu prefixul `glut`.

Dezvoltarea programelor grafice folosind utilitarul GLUT

Biblioteca grafică OpenGL conţine funcţii de redare a primitivelor geometrice independente de sistemul de operare, de orice sistem Windows sau de platforma hardware. Ea nu conţine nici o funcţie pentru a crea sau administra ferestre de afişare pe display (windows) şi nici funcţii de citire a evenimentelor de intrare (mouse sau tastatură). Pentru crearea şi administarea ferestrelor de afişare şi a evenimentelor de intrare se pot aborda mai multe soluţii: utilizarea directă a interfeţei Windows (Win32 API), folosirea compilatoarelor sub Windows, care conţin funcţii de acces la ferestre şi evenimente sau folosirea altor instrumente (utilitare) care înglobează interfaţa OpenGL în sistemul de operare respectiv.

Un astfel de utilitar este GLUT, care se compilează şi instalează pentru fiecare tip de sistem Windows. Header-ul `glut.h` trebuie să fie inclus în fişierele aplicaţiei, iar biblioteca `glut.lib` trebuie legată (linkată) cu programul de aplicaţie. În lucrarea de faţă s-a folosit versiunea `glut3.6`, care poate fi preluată din internet (`www.sgi.com/pub/opengl/GLUT`).

Sub GLUT, orice aplicaţie se structurează folosind mai multe funcţii `callback`. O funcţie `callback` este o funcţie care aparţine programului aplicaţiei şi este apelată de un alt program, în acest caz sistemul de operare, la apariţia anumitor evenimente. În GLUT sunt predefinite câteva tipuri de funcţii callback; acestea sunt scrise în aplicaţie şi pointerii lor sunt transmişi la înregistrare sistemului Windows, care le apelează (prin pointerul primit) în momentele necesare ale execuţiei.

Funcţii de control ale ferestrei de afişare

Sunt disponibile cinci funcţii pentru controlul ferestrei de afişare a programului. Aici trebuie să fie precizate diferitele semnificaţii ale termenului suprautilizat "fereastră". În general, în programare se foloseşte termenul fereastră (*window*) pentru orice zonă de afişare controlată pe display; ferestrele sunt gestionate de sistemul de operare (în cazul sistemului Microsoft Windows (95, NT)) sau de subsistemul grafic X Window care se execută sub sistemele de operare de tip Unix.

În aplicaţiile de grafică tridimensională termenul de fereastră de vizualizare (*view plane window*) se referă la regiunea rectangulară care reprezintă intersecţia dintre planul de vizualizare şi volumul de vizualizare şi în care sunt proiectate toate obiectele vizibile ale scenei. Fereastra de vizualizare se afişează într-o poartă de afişare (*viewport*) care se defineşte prin transformarea fereastră-poartă. Poarta de afişare grafică se amplasează într-o fereastră de afişare (*window*) şi legătura dintre poarta de afişare şi fereastra de afişare este realizată diferit, în funcţie de modul de programare folosit (interfaţă Windows sau toolkit).

În progrmele dezvoltate sub GLUT, corelarea dintre poarta de afişare şi fereastra Window este asigurată de următoarele funcţii:

```
void glutInit(int* argc, char** argv);
```

Această funcţie iniţializează GLUT folosind argumentele din linia de comandă; ea trebuie să fie apelată înaintea oricăror alte funcţii GLUT sau OpenGL.

```
void glutInitDisplayMode(unsigned int mode);
```

Specifică caracteristicile de afişare a culorilor şi a bufferului de adâncime şi numărul de buffere de imagine. Parametrul `mode` se obţine prin SAU logic între valorile fiecărei opţiuni. De exemplu,

```
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH)
```

iniţializează afişarea culorilor în modul RGB, cu două buffere de imagine şi buffer de adâncime. Alte valori ale parametrului `mode` sunt: `GLUT_SINGLE` (un singur buffer de imagine), `GLUT_RGBA` (modelul RGBA al culorii), `GLUT_STENCIL` (validare buffer şablon) `GLUT_ACCUM` (validare buffer de acumulare).

```
void glutInitWindowPosition(int x, int y);
```

Specifică locaţia iniţială pe ecran a colţului stânga sus al ferestrei de afişare.

```
void glutInitWindowSize(int width, int heigth);
```

Defineşte dimensiunea iniţială a ferestrei de afişare în număr de pixeli pe lăţime (`width`) şi înălţime (`heigth`) .

```
int glutCreateWindow(char* string);
```

Creează fereastra în care se afişează contextul de redare (poarta) OpenGL şi returnează identificatorul ferestrei. Această fereastră este afişată numai după apelul funcţiei `glutMainLoop()`.

Funcţii callback.

Funcţiile `callback` se definesc în program şi se înregistrează în sistem prin intermediul unor funcţii GLUT. Ele sunt apelate de sistemul de operare atunci când este necesar, în funcţie de evenimentele apărute.

```
glutDisplayFunc(void(*Display)(void));
```

Această funcţie înregistrează funcţia callback `Display` în care se calculează şi se afişează imaginea. Argumentul funcţiei este un pointer la o funcţie fără argumente care nu returnează nici o valoare. Funcţia `Display` (a aplicaţiei) este apelată oridecâte ori este necesară desenarea ferestrei: la iniţializare, la modificarea dimensiunilor ferestrei, sau la apelul explicit al funcţiei `gluPostRedisplay()`.

```
glutReshapeFunc(void(*Reshape)(int w, int h));
```

Înregistrează funcţia callback `Reshape` care este apelată oridecâte ori se modifică dimensiunea ferestrei de afişare. Argumentul este un pointer la funcţia cu numele `Reshape` cu două argumente de tip întreg şi care nu returnează nici o valoare. În această funcţie, programul de aplicaţie trebuie să refacă transformarea fereastră-poartă, dat fiind că fereastra de afişare şi-a modificat dimensiunile.

```
glutKeyboardFunc(void(*Keyboard)(unsigned int key,
                                 int x, int y);
```

Înregistrează funcţia callback `Keyboard` care este apelată atunci când se acţionează o tastă. Parametrul `key` este codul tastei, iar `x` şi `y` sunt coordonatele (relativ la fereastra de afişare) a mouse-ului în momentul acţionării tastei.

```
glutMouseFunc(void(*MouseFunc)(unsigned int button,
                               int state, int x, int y);
```

Înregistrează funcţia callback `MouseFunc` care este apelată atunci când este apăsat sau eliberat un buton al mouse-ului. Parametrul `button` este codul butonului (poate avea una din constantele `GLUT_LEFT_BUTTON`, `GLUT_MIDDLE_BUTTON` sau `GLUT_RIGHT_BUTTON`) . Parametrul state indică apăsarea (`GLUT_DOWN`) sau eliberarea (`GLUT_UP`) al unui buton al mouse-ului. Parametrii x şi y sunt coordonatele relativ la fereastra de afişare a mouse-ului în momentul evenimentului.

Funcţii de execuţie GLUT

Execuţia unui program folosind toolkit-ul GLUT se lansează prin apelul funcţiei `glutMainLoop()`, după ce au fost efectuate toate iniţializările şi înregistrările funcţiilor callback. Această buclă de execuţie poate fi oprită prin închiderea ferestrei aplicaţiei. În cursul execuţiei sunt apelate funcţiile callback în momentele apariţiei diferitelor evenimente. În cursul execuţiei se mai poate executa un proces atunci cînd nu apare nici un eveniment (atunci când coada de evenimente este vidă). Acest proces execută funcţia callback `IdleFunc` înregistrată prin apelul funcţiei `glutIdleFunc(void(*IdleFunc)(void));` Această funcţie este folosită în animaţie.

Generarea obiectelor tridimensionale

Multe programe folosesc modele simple de obiecte tridimensionale pentru a ilustra diferite aspecte ale prelucrărilor grafice. GLUT conţine câteva funcţii care permit redarea unor astfel de obiecte tridimensionale în modul wireframe sau cu suprafeţe pline (*filled*). Fiecare obiect este reprezentat într-un sistem de referinţă local, iar dimensiunea lui poate fi transmisă ca argument al funcţiei respective. Poziţionarea şi orientarea obiectelor în scenă se face în programul de aplicaţie. Exemple de astfel de funcţii sunt:

```
void glutWireCube(GLdouble size);
void glutSolidCube(GLdouble size);
void glutWireSphere(GLdouble radius, GLint slices,
        GLint stacks);
void glutSolidSphere(GLdouble radius, GLint slices,
        GLint stacks);
```

Programele GLUT au un mod specific de organizare, care provine din felul în care sunt definite şi apelate funcţiile callback. Acest mod va fi prezentat la primul exemplu de program OpenGL-GLUT şi va fi reluat apoi cu mici modificări şi în alte exemple.

Caracteristicile bibliotecii OpenGL

În fişierul header ale bibliotecii OpenGL (`gl.h`) sunt definite mai multe constante simbolice care reprezintă diferite stări, variabile sau valori de selecţie a opţiunilor OpenGL. Aceste constante sunt toate scrise cu majuscule şi sunt precedate de prefixul `GL_`. De exemplu, constantele simbolice care definesc valorile `TRUE` şi `FALSE` şi cele care selectează tipul unei primitive grafice sunt:

```
/* Boolean */
#define GL_TRUE                 1
#define GL_FALSE                0
/* BeginMode */
#define GL_POINTS               0x0000
#define GL_LINES                0x0001
#define GL_LINE_LOOP            0x0002
#define GL_LINE_STRIP           0x0003
#define GL_TRIANGLES            0x0004
#define GL_TRIANGLE_STRIP       0x0005
#define GL_TRIANGLE_FAN         0x0006
#define GL_QUADS                0x0007
#define GL_QUAD_STRIP           0x0008
#define GL_POLYGON              0x0009
```

De asemenea, biblioteca OpenGL defineşte propriile tipuri de date, cele mai multe corespunzând tipurilor de date fundamentale ale limbajului C .De exemplu, în `gl.h` sunt definite următoarele tipuri:

```
typedef unsigned int GLenum;
typedef unsigned char GLboolean;
```

```
typedef signed char GLbyte;
typedef unsigned char GLubyte;
typedef short GLshort;
typedef int GLint;
typedef int GLsizei;
typedef unsigned int GLuint;
typedef float GLfloat;
typedef float GLclampf;
typedef double GLdouble;
typedef double GLclampd;
typedef void GLvoid;
```

Pentru a înţelege funcţionarea comenzilor OpenGL, se descriu în continuare cele mai importante dintre caracteristicile OpenGL.

Poarta de afişare OpenGL

Poarta de afişare mai este numită context de redare (*rendering context*), şi este asociată unei ferestre din sistemul Windws. Dacă se programează folosind biblioteca GLUT, corelarea dintre fereastra Windows şi portul OpenGL este asigurată de funcţii ale acestei biblioteci. Dacă nu se foloseşte GLUT, atunci funcţiile bibliotecilor de extensie XGL, WGL sau PGL permit atribuirea unui context de afişare Windows pentru contextul de redare OpenGL şi accesul la contextul de afişare Windows (*device context*). Funcţia OpenGL care defineşte transformarea fereastră-poartă este:

```
void glViewport(GLint x, GLint y,
        GLsizei width, GLsizei height);
```

unde x şi y specifică poziţia colţului stânga-jos al dreptunghiului porţii în fereastra de afişare (window) şi au valorile implicite 0, 0. Parametrii weidth şi heigth specifică lăţimea, respectiv înălţimea, porţii de afişare, dată ca număr de pixeli. Se reaminteşte că transformarea fereastră-poartă este componentă a transformării din sistemul de referinţă normalizat în sistemul de referinţă ecran 3D (secţiunea 4.4).

Un pixel este reprezentat în OpenGL printr-un descriptor care defineşte mai mulţi parametri:

- numărul de biţi/pixel pentru memorarea culorii
- numărul de biţi/pixel pentru memorarea adâncimii
- numărul de buffere de imagine

Bufferul de cadru

Bufferul de cadru (*frame buffer*) conţine toate datele care definesc o imagine şi constă din mai multe secţiuni logice: bufferul de imagine (sau bufferul de culoare), bufferul de adâncime (*Z-buffer*), bufferul şablon (*stencil*), bufferul de acumulare (*acumulation*).

Bufferul de imagine (*color buffer*) în OpenGL poate conţine una sau mai multe secţiuni, în fiecare fiind memorată culoarea pixelilor din poarta de afişare. Redarea imaginilor folosind un singur buffer de imagine este folosită pentru imagini statice, cel mai frecvent în proiectarea grafică (CAD). În generarea interactivă a

imaginilor dinamice, un singur buffer de imagine produce efecte nedorite, care diminuează mult calitatea imagii generate.

Orice cadru de imagine începe cu ştergerea (de fapt, umplerea cu o culoare de fond) a bufferului de imagine. După aceasta sunt generaţi pe rând pixelii care aparţin tuturor elementelor imaginii (linii, puncte, suprafeţe) şi intensităţile de culoare ale acestora sunt înscrise în bufferul de imagine. Pentru trecerea la cadrul următor, trebuie din nou şters bufferul de imagine şi reluată generarea elementelor componente, pentru noua imagine. Chiar dacă ar fi posibilă generarea şi înscrierea în buffer a elementelor imaginii cu o viteză foarte mare (ceea ce este greu de realizat), tot ar exista un interval de timp în care bufferul este şters şi acest lucru este perceput ca o pâlpâire a imaginii. În grafica interactivă timpul necesar pentru înscrierea datelor în buffer este (în cazul cel mai fericit) foarte apropiat de intervalul de schimbare a unui cadru a imaginii (update rate) şi, dacă acest proces are loc simultan cu extragerea datelor din buffer şi afişarea lor pe display, atunci ecranul va prezenta un timp foarte scurt imaginea completă a fiecărui cadru, iar cea mai mare parte din timp ecranul va fi şters sau va conţine imagini parţiale ale cadrului. Tehnica universal folosită pentru redarea imaginilor dinamice (care se schimbă de la un cadru la altul) este tehnica *dublului buffer de imagine*.

În această tehnică există două buffere de imagine: bufferul din faţă (*front*), din care este afişată imaginea pe ecran şi bufferul din spate (*back*), în care se înscriu elementele de imagine generate. Când imaginea unui cadru a fost complet generată, ea poate fi afişată pe ecran printr-o simplă operaţie de comutare între buffere: bufferul spate devine buffer faţă, şi din el urmează să fie afişată imagine cadrului curent, iar bufferul faţă devine buffer spate şi în el urmează să fie generată imaginea noului cadru. Comutarea între bufferele de imagine se poate sincroniza cu cursa de revenire pe verticală a monitorului şi atunci imaginile sunt prezentate continuu, fără să se observe imagini fragmentate sau pâlpâiri.

În OpenGL comutarea bufferelor este efectuată de funcţia `SwapBuffers()`. Această funcţie trebuie să fie apelată la terminarea generării imaginii tuturor obiectelor vizibile în fiecare cadru.

Modul în care se execută comutarea bufferelor depinde de platforma hardware. În lipsa unui accelerator grafic, toate operaţiile OpenGL sunt executate software, bufferul spate este o zonă din memorie principală, iar bufferul faţă este memoria video din adaptorul grafic. La comutarea bufferelor care se produce la apelul funcţiei `SwapBuffers()`, are loc copierea bufferului back (care este implementat ca un bitmap independent de dispozitiv - DIB - device independent bitmap), în bufferul front, care este memoria de ecran a adaptorului grafic.

Definirea şi utilizarea bufferelor componente ale unui buffer de cadru OpenGL este puţin mai complexă decât modul general de execuţie descris mai sus, deoarece se oferă posibilitatea creerii atât a imaginilor cu simplu sau dublu buffer, monografice şi stereografice şi selectarea bufferului în care se scrie sau se citeşte la un moment dat. Funcţia:

```
void glDrawBuffer(GLenum mode);
```

stabileşte bufferul în care se desenează primitivele geometrice. Parametrul `mode` poate lua una din valorile: `GL_NONE`, `GL_FRONT_LEFT`, `GL_FRONT_RIGHT`,

`GL_BACK_LEFT`, `GL_BACK_RIGHT`, `GL_FRONT`, `GL_BACK`, `GL_LEFT`, `GL_RIGHT`, `GL_FRONT_AND_BACK`. Valoarea implicită este `GL_FRONT` pentru imagini cu un singur buffer şi `GL_BACK` pentru imagini cu dublu buffer. Celelate opţiuni se referă la imagini stereoscopice.

Funcţia `glReadBuffer(GLenum mode)` stabileşte bufferul curent din care se citesc pixelii sursă în operaţii de combinare sau acumulare. Parametrul `mode` poate lua una din valorile descrise mai sus. Valoarea implicită este `GL_FRONT` pentru imagini monografice cu un singur buffer de culoare sau `GL_BACK` pentru imagini monografice cu două buffere de culoare.

Pentru imaginile stereografice, biblioteca OpenGL pune la dispoziţie bufferele necesare pentru definirea celor două imagini diferite (stânga, dreapta), pentru simplu şi dublu buffer şi funcţii pentru selecţia între acestea. Crearea propriu-zisă a imaginilor stereoscopice este rezolvată la nivelul programului de aplicaţie şi necesită dispozitive de afişare adecvate.

Bufferul de adâncime (*depth buffer*) memorează adâncimea fiecărui pixel şi, prin aceasta, permite eliminarea suprafeţelor ascunse. Bufferul de adâncime conţine acelaşi număr de locaţii ca şi un buffer de imagine, fiecare locaţie corespunzând unui pixel, de o anumită adresă.. Valoarea memorată în locaţia corespunzătoare unui pixel este distanţa faţă de punctul de observare (adâncimea pixelului). La generarea unui nou pixel cu aceeaşi adresă, se compară adâncimea noului pixel cu adâncimea memorată în bufferul de adâncime, şi noul pixel înlocuieşte vechiul pixel (îl "ascunde") dacă este mai apropiat de punctul de observare. Bufferul de adâncime se mai numeşte şi Z-buffer, de la coordonata z, care reprezintă adâncimea în sistemul de referinţă ecran 3D.

Bufferul şablon (*stencil buffer*) este un buffer auxiliar care este utilizat pentru restricţionarea desenării în anumite zone ale porţii de afişare. Se pot obţine astfel imagini încadrate în anumite şabloane, ca de exemplu, instrumente de afişare folosite în simulatoare de antrenament. Bufferul şablon poate fi folosit şi pentru redarea suprafeţelor coplanare.

Bufferul de acumulare (*accumulation buffer*) este folosit pentru crearea imaginilor antialising prin acumularea intensităţilor de culoare a pixelilor rezultaţi prin eşantionarea imaginii.

Operațiile de bază OpenGL

OpenGL desenează primitive geometrice (puncte, linii şi poligoane) în diferite moduri selectabile. Primitivele sunt definite printr-un grup de unul sau mai multe vârfuri (*vertices*). Un vârf defineşte un punct, capătul unei linii sau vârful unui poligon. Fiecare vârf are asociat un set de date:

- coordonate,
- culoare,
- normală,
- coordonate de textură.

Aceste date sunt prelucrate independent, în ordine şi în acelaşi mod pentru toate primitivele geometrice. Singura deosebire care apare este aceea că, dacă o linie sau o suprafaţă este decupată, atunci grupul de vârfuri care descriau iniţial primitiva respectivă este înlocuit cu un alt grup, în care pot apare vârfuri noi rezultate din intersecţia laturilor cu planele volumului de decupare (volumul canonic) sau unele vârfuri din cele iniţiale pot să dispară.

Comenzile sunt prelucrate în ordinea în care au fost primite în OpenGL, adică orice comandă este complet executată înainte ca să fie executată o nouă comandă.

OpenGL este o interfaţă procedurală, secvenţa de comenzi (apeluri de funcţii) prescriu ce operaţii să fie executate şi nu ce rezultat s-ar dori să fie obţinut. Se pot specifica valori ale matricelor de transformare, tipul proiecţiei, coeficienţi de reflexie, etc. Datorită naturii procedurale a bibliotecii OpenGL, este important să fie cunoscut modul în care operază, pentru a putea să fie folosită cu succes.

Modelul de execuţie OpenGL este modelul *client-server*. O aplicaţie (client) generează comenzi care sunt interpretate şi executate de către OpenGL (server). Serverul OpenGL poate să fie execut pe un alt calculator decât aplicaţia, dar atunci trebuie să fie definit un protocol de comunicaţie client-server, dat fiind că nu există comenzi OpenGL pentru astfel de operaţii.

Efectul comenzilor OpenGL asupra bufferului de imagine este controlat de sistemul Windows care alocă zona de afişare a porţii (viewport) OpenGL pe ecran.

Figura 6.1 prezintă o diagramă bloc simplificată a modului în care OpenGL prelucrează datele. S-au ignorat pentru început aspectele legate de texturare şi anti-alising. Secvenţa de operaţii OpenGL (numită *pipeline grafic*) se execută asupra vârfurilor şi asupra datelor asociate acestora: coordonate, culoare, normală (şi coordonate de textură, care nu sunt reprezentate în figură). Succesiunea operaţiilor OpenGL urmează aproape identic succesiunea generică a operaţiilor grafice prezentată în Fig. 4.1.

Biblioteca OpenGL primeşte o succesiune de primitive geometrice pe care le desenează, adică le converteşte în mulţimea de pixeli corespunzătoare, înscriind valorile culorilor acestora într-un buffer de imagine. În continuare sunt detaliate fiecare dintre operaţiile grafice prezentate în figură.

Modul în care este executată secvenţa de operaţii pentru redarea primitivelor grafice depinde de starea bibliotecii OpenGL, stare care este definită prin mai multe variabile de stare ale acesteia (parametri). Numărul de variabile de stare ale bibliotecii este destul de mare, descrierea lor ele poate fi găsită în manualul de referinţă (*OpenGL Reference Manual*), iar pe parcursul expunerii vor fi prezentate numai cele mai importante dintre acestea.

La iniţializare, fiecare variabilă de stare este setată la o valoare implicită. O stare o dată setată îşi menţine valoarea neschimbată până la o nouă setare a acesteia. Variabilele de stare au denumiri sub formă de constante simbolice care pot fi folosite pentru aflarea valorilor acestora. De exemplu, în Fig. 6.1 sunt reprezentate două variabile de stare, culoarea curentă şi normala curentă, care au denumirile definite în `gl.h`:

```
#define GL_CURRENT_COLOR            0x0B00
```

```
#define GL_CURRENT_NORMAL               0x0B02
```

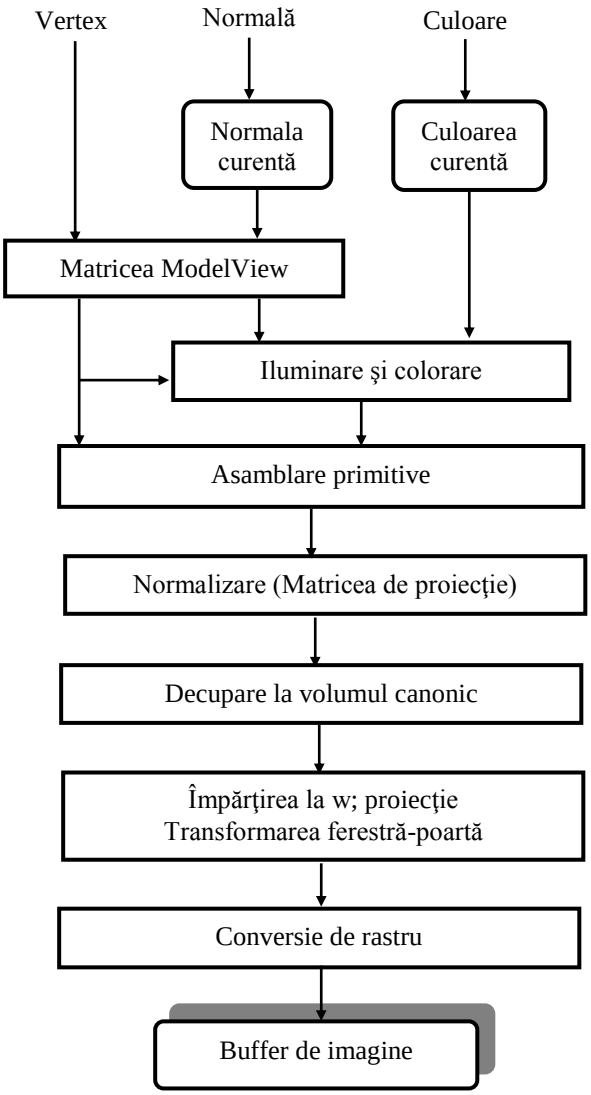


Fig. 6.1 Operaţiile grafice OpenGL

Variabilele de stare OpenGL sunt de două categorii: variabile de tip binar şi variabile definite prin diferite structuri de date.

Variabile de tip binar pot avea una din două stări: starea activă (*enabled*) sau starea inactivă (*disabled*). Setarea la starea activă se realizează prin apelul funcţiei:

```
void glEnable(GLenum param);
```

unde `param` este numele simbolic al parametrului (variabilei de stare).

Setarea la starea inactivă se realizează prin apelul funcţiei:

```
void glDisable(GLenum param);
```

De exemplu, apelul funcţiei `glEnable(GL_DEPTH_TEST)` activează testul de adâncime şi actualizarea corespunzătoare a bufferului de adâncime (*depth buffer*), iar apelul funcţiei `glDisable(GL_DEPTH_TEST)` dezactivează testul de adâncime.

În orice loc într-un program OpenGL, se poate afla valoarea unui parametru binar prin apelul funcţiei:

```
GLboolean glIsEnabled(GLenum param);
```

care returnează `GL_TRUE` dacă parametrul `param` este în starea activă, şi `GL_FALSE`, dacă parametrul `param` este în starea inactivă.

Valoarea unei variabile de stare care nu este de tip binar se setează prin apelul unei funcţii specifice variabilei respective, care are ca argumente valorile necesare pentru actualizare. De exemplu, variabila de stare culoare curentă, denumită `GL_CURRENT_COLOR`, se setează prin funcţia:

```
void glColor3f(GLfloat red, GLfloat green, GLfloat blue);
```

la o valoare dată prin trei componente: roşu (*red*), verde (*green*), albastru (*blue*).

Fiind o bibliotecă dezvoltată în limbajul C, fără posibilitatea de supraîncărcare a funcţiilor, selecţia unei funcţii apelate cu diferite tipuri de argumente de apel este realizată prin modificarea (printr-un sufix) a numelui funcţiei. De exemplu, funcţia de setare a culorii curente are mai multe variante, după tipul şi numărul argumentelor: `glColor3f()`, `glColor3ub()`, `glColor4d()`, etc. În continuare, în această lucrare se notează generic cu # sufixul dintr-o familie de funcţii (de exemplu, `glColor#()`).

Valoarea unui parametru (variabilă de stare) OpenGL care nu este de tip binar se poate obţine în orice loc în program prin apelul uneia din funcţiile `glGet#()`:

```
void glGetBooleanv(GLenum pname, GLboolean* params);
void glGetDoublev(GLenum pname, GLdouble* params);
void glGetFloatv(GLenum pname, GLfloat* params);
void glGetIntegerv(GLenum pname, GLint* params);
```

unde `params` este adreasa unde se depun valorile componente ale parametrului `pname`. De exemplu, prin instrucţiunile:

```
GLfloat color[4];
glGetFloatv(GL_CURRENT_COLOR, color);
```

se obţin cele patru componente ale culorii curente (red, green, blue, alpha) în vectorul `color`.

Primitive geometrice

OpenGL execută secvenţa de operaţii grafice asupra fiecărei primitive geometrice, definită printr-o mulţime de vârfuri şi tipul acesteia. Coordonatele unui

vârf sunt transmise către OpenGL prin apelul unei funcţii `glVertex#()`. Aceasta are mai multe variante, după numărul şi tipul argumentelor. Iată, de exemplu, numai câteva din prototipurile funcţiilor `glVertex#()`:

```
void glVertex2d(GLdouble x, GLdouble y);
void glVertex3d(GLdouble x, GLdouble y, GLdouble z);
void glVertex3f(GLfloat x, GLfloat y, GLfloat z);
void glVertex4d(GLdouble x, GLdouble y,
        GLdouble z, GLdouble w);
```

Vârfurile pot fi specificate în plan, în spaţiu sau în coordonate omogene, folosind apelul funcţiei corespunzătoare.

O primitivă geometrică se defineşte printr-o mulţime de vârfuri (care dau descrierea geometrică a primitivei) şi printr-unul din tipurile prestabilite, care indică topologia, adică modul în care sunt conectate vârfurile între ele. Mulţimea de vârfuri este delimitată între funcţiile `glBegin()` şi `glEnd()`. Aceeaşi mulţime de vârfuri (v_0, v_1, v_2,….v_{n-1}) poate fi tratată ca puncte izolate, linii, poligon, etc, în funcţie de tipul primitivei, care este transmis ca argument al funcţiei `glBegin()`:

```
void glBegin(GLenum mode);
```

Valorile argumentului `mode` şi, deci, tipurile de primitive acceptate de OpenGL sunt prezentate în tabelul de mai jos. În Fig. 6.2 sunt ilustrate aceste primitive geometrice.

Tabelul 6.1 Tipurile de primitive geometrice

Argument	Primitivă geometrică
`GL_POINTS`	Desenează n puncte
`GL_LINES`	Desenează segmentele de dreaptă izolate (v_0,v_1), (v_2,v_3),… ş.a.m.d. Dacă n este impar ultimul vârf este ignorat
`GL_LINE_STRIP`	Desenează linia poligonală formată din segmentele (v_0,v_1), (v_1,v_2), .. . …(v_{n-2}, v_{n-1})
`GL_LINE_LOOP`	La fel ca primitiva `GL_LINE_STRIP`, dar se mai desenează segmentul (v_n, v_0) care închide o buclă.
`GL_TRIANGLES`	Desenează o serie de triunghiuri folosind vârfurile (v_0,v_1,v_2), (v_3,v_4,v_5), ş.a.m.d. Dacă n nu este multiplu de 3, atunci ultimele 1 sau 2 vârfuri sunt ignorate.
`GL_TRIANGLE_STRIP`	Desenează o serie de triunghiuri folosind vârfurile (v_0,v_1,v_2), (v_2,v_1,v_3),..ş.a.m.d. Ordinea este aleasă astfel ca triunghiurile să aibă aceeaşi orientare, deci să poată forma o suprafaţă închisă.
`GL_TRIANGLE_FAN`	Desenează triunghiurile (v_0,v_1,v_2), (v_0,v_2,v_3), ş.a.m.d.
`GL_QUADS`	Desenează o serie de patrulatere (v_0,v_1,v_2,v_3), (v_4,v_5,v_6,v_7), ş.a.m.d. Dacă n nu este multiplu de 4, ultimele 1, 2 sau 3 vârfuri sunt ignorate.
`GL_QUADS_STRIP`	Desenează o serie de patrulatere (v_0,v_1,v_3,v_2), (v_3,v_2,v_5,v_4), ş.a.m.d Dacă n < 4, nu se desenază nimic. Dacă n este impar, ultimul vârf este ignorat.
`GL_POLYGON`	Desenează un poligon cu n vârfuri, (v_0,v_1,…v_{n-1}). Dacă

poligonul nu este convex, rezultatul este impredictibil.

Acest tabel, împreună cu figura de mai jos (Fig. 6.2) sunt suficiente pentru înţelegerea modului în care OpenGL interpretează primitivele geometrice.

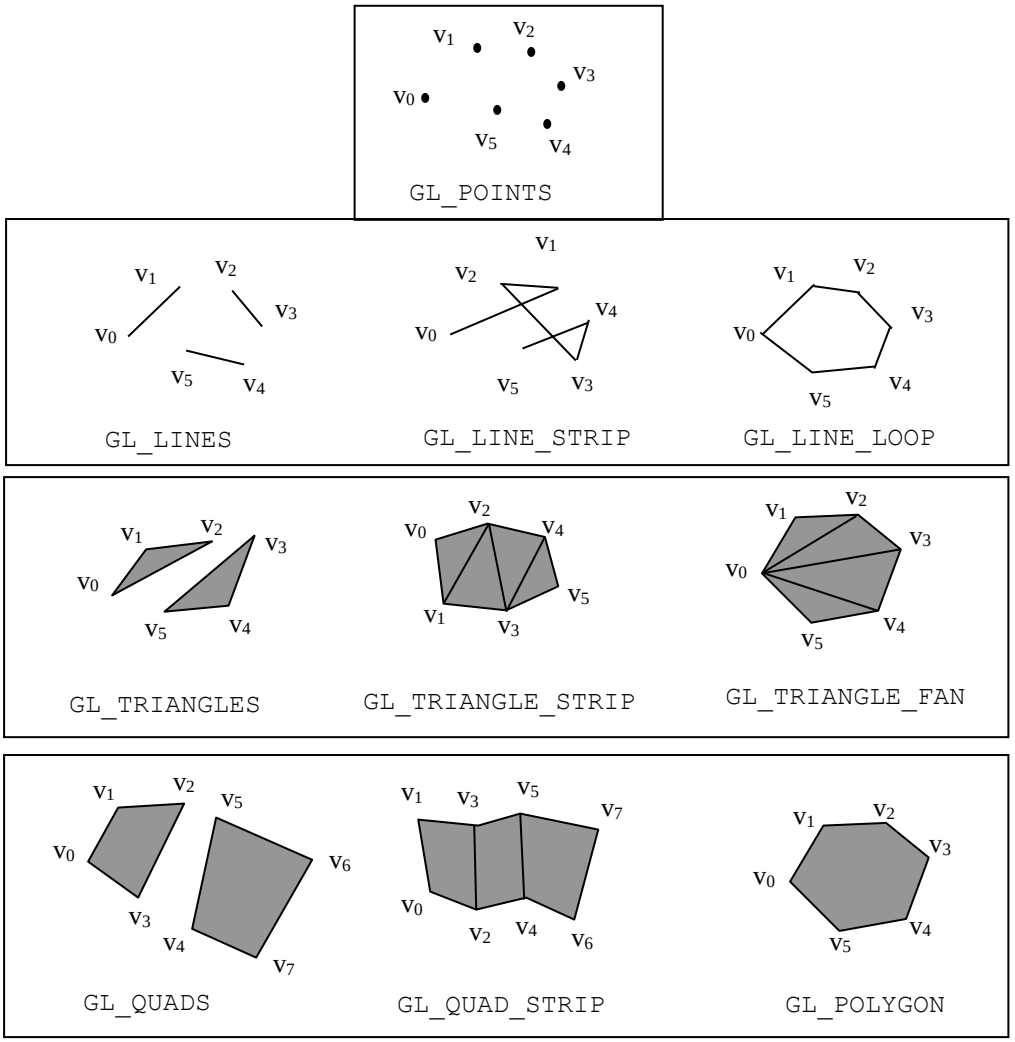


Fig. 6.2 Primitivele geometrice OpenGL

Primitivele de tip suprafaţă (triunghiuri, patrulatere, poligoane) pot fi desenate în modul "cadru de sârmă" (*wireframe*), sau în modul plin (*fill*), prin setarea variabilei de stare `GL_POLYGON_MODE` folosind funcţia:

```
void glPolygonMode(GLenum face, GLenum mode);
```

unde argumentul `mode` poate lua una din valorile:

- `GL_POINT` : se desenează numai vârfurile primitivei, ca puncte în spaţiu, indiferent de tipul acesteia.
- `GL_LINE`: muchiile poligoanelor se desenează ca segmente de dreaptă.

- `GL_FILL`: se desenează poligonul plin.

Argumentul `face` se referă la tipul primitivei geometrice (din punct de vedere al orientării) căreia i se aplică modul de redare `mode`. Din punct de vedere al orientării, OpenGL admite primitive orientate direct (`frontface`) şi primitive orientate invers (`backface`). Argumentul `face` poate lua una din valorile: `GL_FRONT`, `GL_BACK` sau `GL_FRONT_AND_BACK`, pentru a se specifica primitive orientate direct, primitive orientate invers şi, respectiv ambele tipuri de primitive.

În mod implicit, sunt considerate orientate direct suprafeţele ale căror vârfuri sunt parcurse în ordinea inversă acelor de ceas. Acestă setare se poate modifica prin funcţia `glFrontFace(GLenum mode)`, unde `mode` poate lua valoarea `GL_CCW`, pentru orientare în sens invers acelor de ceas (*counterclockwise*) sau `GL_CW`, pentru orientare în sensul acelor de ceasornic (*clockwise*).

Reprezentarea culorilor în OpenGL

În biblioteca OpenGL sunt definite două modele de culori: modelul de culori RGBA şi modelul de culori indexate. În modelul RGBA sunt memorate componentele de culoare R, G, B şi transparenţa A pentru fiecare primitivă geometrică sau pixel al imaginii. În modelul de culori indexate, culoarea primitivelor geometrice sau a pixelilor este reprezentată printr-un index într-o tabelă de culori (*color map*), care are memorate pentru fiecare intrare (index) componentele corespunzătoare R,G,B,A ale culorii. În modul de culori indexate numărul de culori afişabile simultan este limitat de dimensiunea tabelei culorilor şi, în plus, nu se pot efectua unele dintre prelucrările grafice importante (cum sunt umbrirea, antialiasing, ceaţa). De aceea, modelul de culori indexate este folosit în principal în aplicaţii de proiectare grafică (CAD) în care este necesar un număr mic de culori şi nu se folosesc umbrirea, ceaţa, etc.

În aplicaţiile de realitate virtuală nu se poate folosi modelul de culori indexate, de aceea în continuare nu vor mai fi prezentate comenzile sau opţiunile care se referă la acest model şi toate descrierile consideră numai modelul RGBA.

Culoarea care se atribuie unui pixel dintr-o primitivă geometrică depinde de mai multe condiţii, putând fi o culoare constantă a primitivei, o culoare calculată prin interpolare între culorile vârfurilor primitivei, sau o culoare calculată în funcţie de iluminare, anti-aliasing şi texturare. Presupunând pentru moment culoarea constantă a unei primitive, aceasta se obţine prin setarea unei variabile de stare a bibliotecii, variabila de culoare curentă (`GL_CURRENT_COLOR`). Culoarea curentă se setează folosind una din funcţiile `glColor#()`, care are mai multe variante, în funcţie de tipul şi numărul argumentelor. De exemplu, câteva din prototipurile acestei funcţii definite în `gl.h` sunt:

```
void glColor3f(GLfloat red, GLfloat green, GLfloat blue);
void glColor3ub(GLubyte red,GLubyte green, GLubyte blue);
void glColor4d(GLdouble red, GLdouble green,
        GLdouble blue, GLdouble alpha);
```

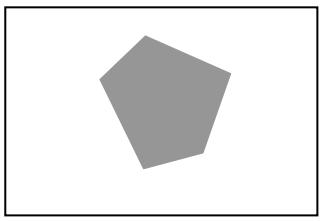
Culoarea se poate specifica prin trei sau patru valori, care corespund componentelor roşu (*red*), verde (*green*), albastru (*blue*), respectiv transparenţă (*alpha*) ca a patra componentă pentru funcţiile cu 4 argumente. Fiecare componentă a culorii curente este memorată ca un număr în virgulă flotantă cuprins în intervalul [0,1]. Valorile argumentelor de tip întreg fără semn (unsigned int, unsigned char, etc.) sunt convertite liniar în numere în virgulă flotantă, astfel încât valoarea 0 este transformată în 0.0, iar valoarea maximă reprezentabilă este transformată în 1.0 (intensitate maximă). Valorile argumentelor de tip întreg cu semn sunt convertite liniar în numere în virgulă flotantă, astfel încât valoarea negativă maximă este transformată în –1.0, iar valoarea pozitivă maximă este transformată în 1.0 (intensitate maximă).

Valoarea finală a culorii unui pixel, rezultată din toate calculele de umbrire, anti-aliasing, texturare, etc, este memorată în bufferul de imagine, o componentă fiind reprezentată printr-un număr `n` de biţi (nu neapărat acelaşi număr de biţi pentru fiecare componentă). Deci componentele culorilor pixelilor memorate în bufferul de imagine sunt numere întreg în intervalul $(0 , 2^n - 1)$, care se obţin prin conversia numerelor în virgulă mobilă prin care sunt reprezentate şi prelucrate culorile primitivelor geometrice, ale materialelor, etc.

ν **Exemplul 6.1**

Secvenţa de funcţii pentru redarea unui poligon în spaţiul tridimensional şi imaginea corespunzătoare generată de OpenGL arată în felul următor:

```
glColor3f(0.8,0.8,0.8);
glBegin(GL_POLYGON);
        glVertex3d(-4,2,-20);
        glVertex3d(-1,5,-20);
        glVertex3d(5,2.5,-20);
        glVertex3d(3,-3,-20);
        glVertex3d(-1,-4,-20);
glEnd();
```



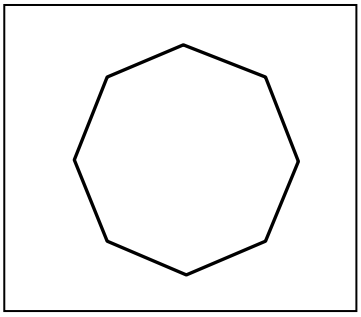
Indentarea instrucţiunilor cuprinse între `glBegin()` şi `glEnd()` nu este necesară în limbajul C (C++) în care este scris codul, ea este folosită doar pentru a marca începutul şi sfârşitul primitivei geometrice. ν

În mod implicit, calculul transparenţei nu este validat. Apelul funcţiei `glEnable(GL_BLEND)`, validează combinarea culorilor, prin care culoarea fiecărui pixel al poligonului este combinată cu culoarea pixelului existent în bufferul de imagine. Modul în care are loc combinarea se setează prin funcţia `glBlendFunc()`, care selectează unul din mai multe moduri de combinare liniară între culoarea primitivei şi culoarea existentă în buffer. Prin combinarea culorilor se simulează transparenţa suprafeţelor precum şi alte tehnici de redare realistă a imaginilor (anti-aliasing, ceaţă). Exemple de transparenţă obţinută prin combinarea culorilor sunt date în secţiunea 7.4.5.1.

ν **Exemplul 6.2**

Se poate genera un poligon regulat prin calculul coordonatelor vârfurilor şi folosirea lor ca argumente ale funcţiilor `glVertex#()`. De exemplu, calculul şi desenarea în OpenGL a conturului unui octogon în spaţiu este următoarea:

```
#define PI 3.141592
int n = 8;
double radius = 10;
glColor3d(0,0,0);
glBegin(GL_LINE_LOOP);
        for (int i=0;i<n;i++){
                double angle = 2*PI*i/n;
                glVertex3d(radius*cos(angle),
        radius*sin(angle),-20);
        }
glEnd();
```



ν

Sistemul de vizualizare OpenGL

Sistemul de vizualizare OpenGL defineşte sistemele de referinţă, transformările geometrice şi relaţiile (matriceale) de transformare pentru redarea primitivelor geometrice. Sistemul de vizualizare OpenGL este o particularizare a sistemului PHIGS, în care centrul sistemului de referinţă de observare (VRP) coincide cu centrul de proiecţie (PRP).

Sistemele de referinţă

Sistemul de referinţă universal definit în OpenGL este un sistem drept, iar matricea care reprezintă un punct (în plan, în spaţiul tridimensional sau în coordonate omogene) este matrice coloană.

Pentru reprezentarea unei matrice 4x4 folosite pentru transformări geometrice în coordonate omogene, în OpenGL este utilizată convenţia coloană majoră: fiind dat un număr de 16 valori reale (tip `float` sau `double`), în ordinea a_0, a_1, a_2,… a_{15}, matricea corespunzătoare este:

$$\mathbf{A} = \begin{bmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{bmatrix}$$

Aceste convenţii sunt aceleaşi cu convenţiile adoptate în lucrarea de faţă, deci matricele de transformări geometrice ale bibliotecii OpenGL sunt identice cu cele prezentate în secţiunile precedente.

Sistemul de referinţă de observare este un sistem de referinţă drept, definit ca localizare şi orientare relativ la sistemul de referinţă universal. În acest sistem de referinţă de observare se poate defini o transformare de proiecţie paralelă ortografică sau de proiecţie perspectivă.

Funcţiile oferite de OpenGL permit definirea proiecţiei perspective pe un plan perpendicular pe axa z a sistemului de referinţă de observare, cu direcţia de observare spre –z (Fig. 6.3). Trunchiul de piramidă de vizualizare este orientat spre $-z_V$ şi poate fi definit prin valorile:

- *znear* : distanţa (dată ca valoare pozitivă) a planului de vizualizare şi a planului de vizibilitate apropiat faţă de centrul sistemului de referinţă de observare (O_V)
- *zfar* : distanţa (dată ca valoare pozitivă) a planului de vizibilitate depărtat faţă de centrul sistemului de referinţă de observare
- *left*, *right*: specifică coordonatele planelor verticale ale volumului de vizualizare
- *top*, *bottom*: specifică coordonatele planelor orizontale ale volumului de vizualizare.

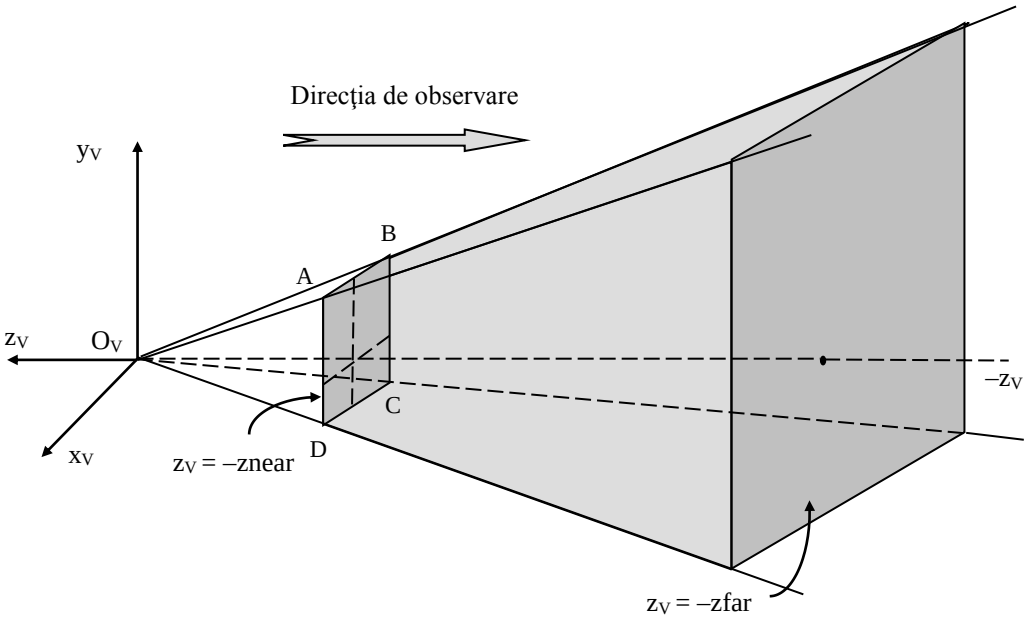


Fig. 6.3 Definirea proiecţiei perspectivă şi a volumului de vizualizare în OpenGL.

Colţul din stânga-jos (D) al ferestrei de vizualizare are coordonatele (`left`, `bottom`, `-znear`) în sistemul de referinţă de observare, iar colţul din dreapta-sus (B) are coordonatele (`right`, `top`, `-znear`) în acelaşi sistem.

Funcţia OpenGL care defineşte o astfel de proiecţie perspectivă este funcţia `glFrustum()`, care are prototipul:

```
void glFrustum(GLdouble left, GLdouble right,
        GLdouble bottom, GLdouble top,
        GLdouble znear, GLdouble zfar);
```

Matricea de normalizare calculată în acest caz este o particularizare a produsului matricelor date de relaţiile 4.14, 4.15, 4.16, în care:

`n = -znear; d = -znear; f = -zfar;`
x_{min} = left; x_{max} = right; y_{min} = bottom; y_{max} = top.

Deducerea ei este propusă ca un exerciţiu pentru cursanţi.

Transformări geometrice

Aşa după cum s-a mai arătat, nu este eficient să se calculeze produsul dintre matricea de reprezentare a fiecărui punct şi matricele de transformări succesive, ci se calculează o matrice de transformare compusă, care se poate aplica unuia sau mai multor obiecte. Pentru calculul matricelor de transformare compuse (prin produs de matrice) se foloseşte o *matrice de transformare curentă* şi operaţii de acumulare în matricea curentă prin postmultiplicare (înmulţire la dreapta): se înmulţeşte matricea curentă cu noua matrice (în această ordine) şi rezultatul înlocuieşte conţinutul matricei curente.

Pentru secvenţa de transformări în ordinea $\mathbf{M_1, M_2 \ldots M_n}$, aplicate obiectelor, se iniţializează matricea curentă **C** cu matricea identitate, după care se calculează matricea compusă prin postmultiplicare în ordine inversă celei în care se aplică matricele componente:

$$\mathbf{C = I}$$
$$\mathbf{C = C M_n = M_n}$$
$$\mathbf{C = C M_{n-1} = M_n M_{n-1}}$$
$$\mathbf{\ldots\ldots\ldots\ldots\ldots\ldots}$$
$$\mathbf{C = C M_2 = M_n M_{n-1} \ldots\ldots M_2}$$
$$\mathbf{C = C M_1 = M_n M_{n-1} \ldots\ldots M_2 M_1}$$
$$\mathbf{C = M = M_n M_{n-1} \ldots\ldots M_2 M_1}$$
$$\mathbf{P' = MP = M_n M_{n-1} \ldots\ldots M_2 M_1 P}$$

În această transformare, se aplică mai întâi matricea $\mathbf{M_1}$ punctului **P**; punctului transformat rezultat i se aplică transformarea $\mathbf{M_2}$, ş.a.m.d.

Se poate reţine deci că matricea compusă **M** se scrie ca produs în ordinea inversă aplicării transformărilor componente şi tot în ordine inversă se acumulează în matricea curentă.

Acest mod de calcul al matricelor compuse are suport în OpenGL prin mai multe variabile interne (matrice curente) şi funcţii prin care acestea pot fi calculate.

Pentru început, se consideră o matrice curentă oarecare **C** stabilită în OpenGL printr-o comandă corespunzătoare (va fi prezentată mai jos).

Matricea curentă se poate iniţializa cu matricea identitate prin funcţia `glLoadIdentity()`, sau cu o matrice oarecare, dată prin pointer la un vector de 16 valori consecutive, prin funcţia `glLoadMatrix#()`:

```
glLoadMatrixd(const GLdouble* m);
glLoadMatrixf(const GLfloat* m);
```

Valorile din vectorul `GLdouble* m` (respectiv `GLfloat* m`) sunt atribuite în ordinea coloană majoră matricei curente.

Conţinutul matricei curente poate fi modificat prin multiplicare (la dreapta) cu o altă matrice, dată printr-un vector de 16 valori de tip `double` sau `float` utilizând funcţia `glMultMatrix#()`:

```
glMultMatrixd(const GLdouble* m);
glMultMatrixf(const GLfloat* m);
```

Matricea curentă **C** este înlocuită cu produsul **C M**, unde **M** este matricea corespunzătoare vectorului dat prin pointerul `m`.

Crearea unei matrice pentru o transformare elementară (translaţie, scalare, etc) şi înmulţirea ei cu matricea curentă se poate face prin apelul unei singure funcţii OpenGL.

Transformarea de translaţie cu valorile x, y, z se implementează prin apelul uneia din funcţiile `glTranslated()` sau `glTranslatef()`, după tipul argumentelor de apel:

```
glTranslated(GLdouble x, GLdouble y, GLdouble z);
glTranslatef(GLfloat x, GLfloat y, GLfloat z);
```

Funcţia `glTranslate#()` creează o matrice de translaţie **T**(x,y,z), dată de relaţia 3.2, înmulţeşte la dreapta matricea curentă **C**, iar rezultatul înlocuieşte conţinutul matricei curente **C**, deci: **C** = **C T**(x,y,z).

Transformarea de scalare este efectuată de una din funcţiile `glScaled()` sau `glScalef()`:

```
glScaled(GLdouble x, GLdouble y, GLdouble z);
glScalef(GLfloat x, GLfloat y, GLfloat z);
```

Funcţia `glScale#()` crează o matrice de scalare **S**(x,y,z), şi o înmulţeşte cu matricea curentă, rezultatul fiind depus în matricea curentă.

Transformarea de rotaţie se defineşte printr-o direcţie de rotaţie dată prin vectorul de poziţie x,y,z şi un unghi `angle` (specificat în grade). Prototipurile funcţiilor de rotaţie sunt:

```
glRotated(GLdouble angle, GLdouble x,
      GLdouble y, GLdouble z);
glRotatef(GLfloat angle, GLfloat x,
      GLfloat y, GLfloat z);
```

Rotaţiile în raport cu axele de coordonate sunt cazuri particulare ale funcţiilor `glRotate#()`. De exemplu, rotaţia cu unghiul `angle` în raport cu axa `x` se obţine la apelul funcţiei `glRotated(angle,1,0,0)` sau `glRotatef(angle,1,0,0)`.

Transformările compuse se efectuează prin acumularea produsului matricelor componente în matricea curentă. De exemplu, transformările de modelare din Exemplul 3.3 au fost implementate în OpenGL după cum urmează.

Transformarea $\mathbf{M_2} = \mathbf{T}(0,-8,0)\ \mathbf{S}(2,2,2)$:

```
glTranslated(0,-8,0);
glScaled(2,2,2);
```

Transformarea $\mathbf{M_3} = \mathbf{T}(8,0,0)\mathbf{R_Z}(\pi/4)\mathbf{S}(2,1,2)$:

```
glTranslated(8,0,0);
glRotated(45,0,0,1);
glScaled(2,1,2);
```

Transformarea $\mathbf{M_4} = \mathbf{T}(-8,0,0)\mathbf{R_Z}(-\pi/4)\mathbf{S}(2,1,2)$:

```
glTranslated(-8,0,0);
glRotated(-45,0,0,1);
glScaled(2,1,2);
```

Stivele matricelor de transformare. După cum s-a mai arătat, transformarea unui obiect din sistemul de referinţă local (de modelare) în sistemul de referinţă normalizat este compusă din succesiunea transformărilor de modelare(instanţiere)$\mathbf{M_I}$, de observare $\mathbf{M_V}$ şi de normalizare $\mathbf{M_N}$:

$$\mathbf{M=M_N M_V M_I}$$

Transformarea de instanţiere $\mathbf{M_I}$ este, în general, specifică fiecărui obiect, deci se calculează pentru fiecare obiect în parte.

Pentru cele mai multe din aplicaţiile grafice, în care scena este observată dintr-un singur punct de observare, matricea de observare $\mathbf{M_V}$ este unică pentru toate obiectele într-un cadru dat, dar se modifică la cadrul următor (imaginea următoare), dacă observatorul şi-a schimbat poziţia de observare, situaţie cea mai probabilă în grafica interactivă.

Matricea de normalizare $\mathbf{M_N}$ este definită de parametrii de proiecţie, care, aşa cum s-a prezentat în secţiunea precedentă, corespund construcţiei sistemului grafic (tipul şi numărul de proiectoare, unghiul de vizibilitate) deci această matrice este o caracteristică constructivă a sistemului grafic şi rămîne constantă pe toată perioada desfăşurării programului, pentru toate obiectele şi pentru toate imaginile generate (cadrele de imagine). În concluzie, sunt necesare următoarele matrice de transformare în cursul generării imaginilor succesive, la momentele (cadrele) i, i+1, i+2, …

Cadrul i: $\mathbf{M=M_N M_{Vi} M_{I1}}$, pentru obiectul 1
$\mathbf{M=M_N M_{Vi} M_{I2}}$, pentru obiectul 2
……..
$\mathbf{M=M_N M_{Vi} M_{Ik}}$, pentru obiectul k
……

Cadrul i+1: $\mathbf{M=M_N M_{Vi+1} M_{I1}}$, pentru obiectul 1
$\mathbf{M=M_N M_{Vi+1} M_{I2}}$, pentru obiectul 2
……..
$\mathbf{M=M_N M_{Vi+1} M_{Ik}}$, pentru obiectul k
……

Se pune problema care este cea mai eficientă organizare a acestor operaţii de transformări succesive. Dacă s-ar utiliza o singură matrice curentă, secvenţa operaţiilor de compunere ar trebui să fie efectuată pentru fiecare obiect, în fiecare cadru de imagine. Este evident că utilizarea unei singure matrice curente de transformare este ineficientă şi nejustificată, dat fiind că o matrice este o resursă extrem de puţin costisitoare.

Soluţia adoptată în OpenGL este de a se folosi mai multe stive de matrice de transformare, în care să fie reţinute şi reutilizate matricele intermediare de transformare.

Transformările de modelare şi de observare sunt efectuate într-o stivă de matrice, numită stiva matricelor de modelare-vizualizare (*modelview matrix stack*). Transformarea de normalizare este prelucrată într-o stivă de matrice separată, numită stiva matricelor de proiecţie (*projection matrix stack*). Mai există încă o stivă de matrice pentru operaţiile de texturare, numită stiva matricelor de texturare (*texture matrix stack*). Separarea matricelor de modelare-vizualizare şi de proiecţie permite execuţia în paralel a operaţiilor din pipeline-ul grafic al acceleratoarelor hardware care au prevăzute resurse pentru astfel de prelucrări.

Fiecare stivă de matrice se comportă asemănător cu o stivă obişnuită, asupra căreia se operează prin funcţii de introducere (`push`) şi extragere (`pop`). Aceste funcţii diferă foarte puţin faţă de funcţiile `push` şi `pop` folosite în general în programare.

În orice moment, una din cele trei stive de matrice este declarată ca stivă curentă şi toate operaţiile cu matricele de transformare sunt adresate stivei curente. Setarea unei stive curente se face prin apelul funcţiei

```
void glMatrixMode(GLenum mode);
```

unde argumentul `mode` poate lua una din constantele simbolice:

GL_MODELVIEW, pentru setarea stivei de modelare-vizualizare ca stivă curentă
GL_PROJECTION, pentru setarea stivei de proiecţie ca stivă curentă
GL_TEXTURE, pentru setarea stivei de texturare ca stivă curentă

Matricea din vârful stivei curente este matricea curentă **C**, cu care se efectuează toate operaţiile cu matrice(`glLoadIdentity()`, `glLoadMatrix#()`, `glMultMatrix#()`, `glTranslate#()`, etc).

Funcţia:

```
void glPushMatrix();
```

adaugă o nouă matrice în stiva curentă, egală cu matricea din vârful acesteia, care devine noul vârf al stivei (Fig. 6.4). Încărcarea unei noi valori în noua matrice curentă se face prin operaţii ulterioare de încărcare sau acumulare prin postmultiplicare. De fapt, aceasta este caracteristica cea mai importantă a stivelor matricelor de transformări, că operaţia `push` se execută prin două sau mai multe funcţii: `glPushMatrix()` crează o nouă poziţie în capul stivei, identică cu valoarea precedentă, şi în această poziţie se continuă operaţiile de compunere a matricelor prin postmultiplicare.

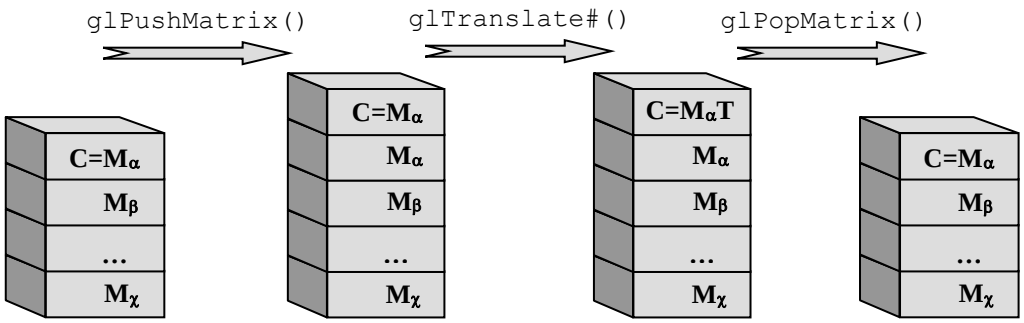


Fig. 6.4 Operaţii cu stiva matricelor de transformare

Funcţia:

```
void glPopMatrix();
```

elimină matricea din vârful stivei curente, iar matricea precedentă acesteia devine noua matrice curentă. Această funcţie nu salvează matricea eliminată (deci ea nu este "extrasă", ca în operaţiile `pop` obişnuite), iar dacă această matrice ar mai fi necesară, ea trebuie să fie copiată în altă zonă de memorie, înainte de apelul funcţiei `glPopMatrix()`.

Funcţia `glPushMatrix()` se apelează atunci când o matrice aflată în capul stivei trebuie să fie salvată pentru a fi utilizată ulterior. Funcţia `glPopMatrix()` se apelează atunci când operaţiile de compunere a matricelor trebuie să continue cu o matrice aflată *sub* capul stivei.

La iniţializarea bibliotecii OpenGL, fiecare stivă de matrice conţine o matrice care este vârful stivei respective.

Se poate prezenta acum un program OpenGL simplu, folosind utilitarul `GLUT`, care exemplifică folosirea stivelor matricelor de transformări geometrice.

ν **Exemplul 6.3**

Se reia ca exemplu de utilizare a stivelor matricelor de transformare programul din exemplul 4.2, în care se generează imaginea unei scene obţinută prin instanţierea a patru obiecte.

Matricea de normalizare şi proiecţie se calculează o singură dată şi se introduce în vârful stivei matricelor de proiecţie. Pentru aceasta, se setează ca stivă curentă stiva matricelor de proiecţie şi se iniţializează matricea din vârful acestei stive cu matricea identitate:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
```

Valoarea matricei de normalizare se setează folosind funcţia `glFrustum()`, descrisă mai sus. Funcţia `glFrustum()` creează matricea de normalizare

corespunzătoare argumentelor de apel, o înmulţeşte cu matricea curentă (care are valoarea de matrice identitate), şi depune rezultatul în matricea curentă, care este matricea din vârful stivel matricelor de proiecţie. În acest exemplu se apelează:

```
glFrustum(–1, 1,–1, 1, 1, 40);
```

Matricele de transformare de modelare şi observare se compun în stiva matricelor de modelare-vizualizare. Sistemul de referinţă de observare are centrul în punctul (`xv, zv, yv`) şi este rotit cu un unghi `gama` în raport cu axa z. Paşii de calcul a matricelor sunt următorii:

(1) Mai întâi se setează ca stivă curentă stiva de modelare-vizualizare şi se iniţilalizează matricea din vârful acesteia cu matricea identitate:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

(2) Se creează în stiva curentă matricea de transformare de observare $\mathbf{M_V} = \mathbf{R_Z}(–\text{gama})\mathbf{T}(–\text{xv}, –\text{yv}, –\text{zv})$:

```
glRotated(-gama, 0,0,1);
glTranslated(-xv,-zv,-zv);
```

(3) Primul obiect are matricea de instanţiere $\mathbf{M_{I1}} = \mathbf{I}$, deci matricea curentă (care conţine matricea de vizualizare) se aplică coordonatelor vârfurilor cubului din sistemul de referinţă de modelare. Funcţia Cube() transmite aceste vârfuri către OpenGL:

```
Cube();
```

(4) Dacă s-ar construi în acest moment matricea de transformare pentru obiectul al doilea, prin multiplicarea matricei curente cu matricele componente de instanţiere din $\mathbf{M_2} = \mathbf{T}(0,–8,0)\ \mathbf{S}(2,2,2)$, atunci matricea de observare, s-ar pierde şi pentru următorul obiect ar trebui calculată din nou. De aceea, se reţine în stivă matricea de observare prin apelul funcţiei `glPushMatrix()`, care crează o nouă matrice în vârful stivei, cu aceeaşi valoare (egală cu $\mathbf{M_V}$). Matricea de instanţiere $\mathbf{M_2}$ se compune cu matricea de observare în această nouă matrice curentă, se apelează funcţia `Cube()`, după care matricea din vârful stivei ($\mathbf{M_V M_2}$) nu mai este necesară. Prin apelul funcţiei `glPopMatrix()`, această matrice este eliminată şi, în acelşi timp, se regăseşte în vârful stivei matricea de observare $\mathbf{M_V}$, care va putea fi folosită pentru obiectele următoare. Funcţiile apelate în acest pas sunt:

```
glPushMatrix();
        glTranslated(0,-8,0);
        glScaled(2,2,2);
        Cube();
glPopMatrix();
```

(5) Pentru celelalte obiecte se repetă operaţiile de la punctul 4.

Programul următor, care conţine cele explicate mai sus, este un exemplu tipic de program grafic folosind biblioteca OpenGL şi utilitarul GLUT. În acest program se poate urmări iniţializarea bibliotecii GLUT, definirea funcţiilor `callback` şi modul de utilizare a stivelor de transformări geometrice. Unele funcţii nou introduse în program sunt explicate în comentarii.

```
#include <gl/glut.h>
// Initializari
void Init(){
	glClearColor(1.0,1.0,1.0,0.0); // culoare de stergere
	glEnable(GL_DEPTH_TEST);		// validare Z-buffer
}
// Redare cub
void Cube(){
	glColor3f(0.8,0.8,0.8);
	glBegin(GL_POLYGON);
		glVertex3d(-1,-1, 1);
		glVertex3d( 1,-1, 1);
		glVertex3d( 1, 1, 1);
		glVertex3d(-1, 1, 1);
	glEnd();
	glColor3f(0,0,0);
	glBegin(GL_LINE_LOOP);
		glVertex3d(-1,-1,-1);
		glVertex3d( 1,-1,-1);
		glVertex3d( 1, 1,-1);
		glVertex3d(-1, 1,-1);
	glEnd();
	glBegin(GL_LINE_LOOP);
		glVertex3d(-1,-1,-1);
		glVertex3d( 1,-1,-1);
		glVertex3d( 1,-1, 1);
		glVertex3d(-1,-1, 1);
	glEnd();
	glBegin(GL_LINE_LOOP);
		glVertex3d(-1,1,-1);
		glVertex3d( 1,1,-1);
		glVertex3d( 1,1, 1);
		glVertex3d(-1,1, 1);
	glEnd();
	glBegin(GL_LINE_LOOP);
		glVertex3d(-1,-1,-1);
		glVertex3d(-1, 1,-1);
		glVertex3d(-1, 1, 1);
		glVertex3d(-1,-1, 1);
	glEnd();
	glBegin(GL_LINE_LOOP);
		glVertex3d(1,-1,-1);
		glVertex3d(1, 1,-1);
		glVertex3d(1, 1, 1);
		glVertex3d(1,-1, 1);
	glEnd();
}
// Generare axe de coordonate
void Axes(double x,double y, double z){
	glColor3f(0,0,0);
```

```
            glBegin(GL_LINES);
                        glVertex3d(0,0,0);
                        glVertex3d(x,0,0);
                        glVertex3d(0,0,0);
                        glVertex3d(0,y,0);
                        glVertex3d(0,0,0);
                        glVertex3d(0,0,z);
            glEnd();
            // Texte x, y, z
            glColor3ub(0, 0, 0);
            glRasterPos3f(1.1*x, 0.0, 0.0);
  glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, 'x');
  glRasterPos3f(0.0, 1.1*y, 0.0);
  glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, 'y');
  glRasterPos3f(0.0, 0.0, 1.1*z);
  glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, 'z');
            // SAGETI
            ……..
}
// Functia callback apelata la schimbarea dim. ferestrei
void Reshape(int w, int h){
   h = (h == 0) ? 1 : h;
   glViewport(0, 0, w, h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
            glFrustum(-(GLfloat)w/(GLfloat)h,
                        (GLfloat)w/(GLfloat)h, -1, 1, 1, 40);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}
static double xv = 2, yv = 2, zv = 20;
static double gama = 0;

// Functia callback de generare imagine
void Display(){
            glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
            glPushMatrix();
                        glRotated(-gama, 0,0,1);
                        glTranslated(-2,-2,-20);
            // transf. de obs.: C = Rz(–30)T(–2,–2,–20)

                        Axes(16,8,16);
                        Cube();                                 // primul obiect

                        glPushMatrix();     // al doilea obiect
                                    glTranslated(0,-8,0);
                                    glScaled(2,2,2);
                                    // C = Rz(–30)T(–2,–2,–20)T(0,–8,0)S(2,2,2)
                                    Cube();
                        glPopMatrix();      // C = Rz(–30)T(–2,–2,–20)
                        glPushMatrix();     // al treilea obiect
                                    glTranslated(8,0,0);
                                    glRotated(45,0,0,1);
                                    glScaled(2,1,2);
                                    // C = Rz(–30)T(–2,–2,–20)T(8,0,0)Rz(45)S(2,2,2)
                                    Cube();
                        glPopMatrix();                  // C = Rz(–30)T(–2,–2,–20)
```

```
                glPushMatrix();      // al patrulea obiect
                        glTranslated(-8,0,0);
                        glRotated(-45,0,0,1);
                        glScaled(2,1,2);
                        //C = Rz(–30)T(–2,–2,–20)T(–8,0,0)Rz(–45)S(2,1,2)
                        Cube();
                glPopMatrix();
        glPopMatrix();
        glutSwapBuffers();
}
int main(int argc, char** argv){
  glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB
                | GLUT_DEPTH );
        glutInitWindowSize(500,500);
        glutInitWindowPosition(100,100);
        glutCreateWindow("Transformari grafice");

        Init();
        glutDisplayFunc(Display);
        glutReshapeFunc(Reshape);
        glutMainLoop();
        return 0;
}
```

În funcţia locală de iniţializare `Init()` se setează culoarea de ştergere a bufferului de imagine folosind funcţia `glClearColor`(). Această culoare de ştergere este memorată într-o variabilă de stare şi este folosită pentru ştergerea bufferului de imagine atunci când se apelează funcţia `glClear()`. De asemenea, se validează testul de adâncime, pentru eliminarea suprafeţelor ascunse.

Funcţia cea mai importantă a programului este funcţia `Display()` în care se execută toate operaţiile de generare a unei imagini: ştergerea bufferului de imagine, (funcţia `glClear()`), crearea matricelor de transformare în stiva matricelor de modelare-vizualizare, generarea imaginii a patru obiecte aplicând transformarea corespunzătoare modelului cub (funcţia `Cube()`). La sfârşitul operaţiilor de generare a imaginii, se comută bufferul de imagine prin apelul funcţiei `glutSwapBuffers()`. Valorea matricei curente în fiecare pas de execuţie este scrisă în comentarii. Indentările instrucţiunilor nu sunt necesare, dar permit urmărirea mai uşoară a operaţiilor cu stivele de matrice. Funcţia callback `Display` este apelată de sistemul de operare ori de câte ori este necesară redesenarea ferestrei de afişare.

O formă foarte simplificată de desenare a unui obiect tridimensional format din mai multe feţe, fiecare faţă formată din mai multe vârfuri, este dată în funcţia `Cube()`. Către OpenGL se transmit primitive geometrice compuse din vârfurile fiecărei feţe date în coordonate în sistemul de referinţă local (de modelare). Dacă se urmăreşte diagrama de execuţie a operaţiilor în OpenGL din Fig. 6.1, se vede că asupra vârfurilor unei primitive se aplică transformarea din matricea curentă din stiva de modelare-vizualizare. De aceea este posibil ca, pentru fiecare instanţiere, să fie transmise aceleaşi coordonate (ale modelului), care sunt transformate (instanţiere şi observare) în sistemul de referinţă de observare. În exemplul dat, prima faţă se desenează ca un poligon plin de culoare gri, iar celelate ca muchii de culoare neagră, pentru o mai bună percepere a imaginii desenate alb-negru.

Secvenţa de operaţii continuă cu asamblarea primitivelor geometrice, folosind informaţiile topologice transmise de funcţia `glBegin()` şi vârfurile primitivei transformate în sistemul de referinţă de observare, normalizarea, folosind matricea din vârful stivei matricelor de proiecţie, decuparea, împărţirea omogenă, proiecţia, transformarea fereastră poartă şi conversia de rastru pentru înscrierea valorilor corespunzătoare a pixelilor primitivei geometrice.

Funcţia de redare a axelor de coordonate `Axes()` generează axele de coordonate şi numele lor, printr-un caracter afişat folosind funcţia: `glutBitmapCharacter()`. Partea de desenare a săgeţilor axelor nu este prezentată dar se poate completa cu uşurinţă.

Funcţia callback de redimensionare `Reshape()` primeşte la apelul ei de către sistem dimensiunile `w` şi `h` ale ferestrei de afişare. În această funcţie se setează dimensiunea porţii egală cu dimensiunea ferestrei de afişare şi cu colţul stânga-jos în punctul de coordonate (0,0) al ferestrei, prin apelul funcţiei `glViewport()`. Tot în această funcţie callback este definită transformarea de normalizare şi proiecţie perspectivă (`glFrustum()`) astfel încât scalarea dintre fereastra de vizualizare şi poarta de afişare (care a fost setată egală cu fereastra de afişare) să fie uniformă, adică raportul dintre dimensiunile pe orizontală şi verticală ale ferestrei de vizualizare să fie egal cu raportul dintre dimensiunile pe orizontală şi verticală ale porţii de afişare. Funcţia de definire a proiecţiei perspectivă şi a volumului de vizualizare, `glFrustum()`, operează asupra stivei matricelor de proiecţie. Funcţia `Reshape()` este apelată atât la lansarea programului aplicaţiei cât şi la orice modificare a dimensiunilor ferestrei.

În funcţia `main()` a programului aplicaţiei sunt apelate funcţiile GLUT de iniţializare a ferestrei de afişare, sunt înregistrate funcţiile callback necesare şi apoi se intră în execuţia programului la apelul funcţiei `glutMainLoop()`.

Imaginile din figurile 4.7 a,b,c,d,e,f s-au obţinut prin modificarea corespunzătoare a valorilor variabilelor `xv`, `yv`, `zv`, `gama` şi recompilarea programului.

ν

Stivele matricelor de transformare sunt deosebit de utile pentru redarea scenelor şi a obiectelor modelate ierarhic, în care scena sau fiecare obiect este compus din mai multe subobiecte, pe mai multe nivele de ierarhie. Fiecare obiect component este definit printr-o matrice de localizare relativ la obiectul căruia îi aparţine (obiect părinte), şi instanţierea acestuia se obţine prin compunerea matricei de localizare proprii cu matricea de localizare a obiectului părinte. Reprezentarea şi redarea scenelor modelate ierarhic este tratată în secţiunea 6, dar în exemplul următor se va prezenta un scurt program de aplicaţie care foloseşte compunerea mai multor transformări de modelare.

ν Exemplul 6.4

Programul descrie construirea unui sistem solar foarte simplificat, cu soarele, două planete şi un satelit al uneia dintre planete. Sferele care reprezintă aceste corpuri sunt redate folosind funcţii ale bibliotecii `GLUT`, `gluSolidSphere()` şi

`gluWireSphere()` care primesc ca argumente dimensiunea, numărul de paralele şi numărul de meridiane ale sferei.

Se consideră că soarele se află în centrul sistemului de referinţă universal şi că planetele execută o mişcare de rotaţie în jurul soarelui în planul z = 0, la distanţe diferite. Satelitul execută o mişcare de rotaţie în jurul unei planete, de asemenea în planul z = 0. Sistemul de referinţă de observare are axele paralele cu axele sistemului universal şi centrul pe axa z pozitivă, la o valoare convenabil aleasă pentru a avea obiectele în câmpul de vizibilitate.

Poziţia centrului unei planete depinde de raza de rotaţie r_P în jurul soarelui şi de unghiul de rotaţie (dat de variabila ρ_P care variază între 0 şi 360°) şi are coordonatele $C_P(r_P \cos\rho_P, r_P \sin\rho_P, 0)$. Se poate observa uşor că aceaste coordonate se obţin printr-o translaţie cu $\mathbf{T}(r_P,0,0)$, urmată de o rotaţie cu $\mathbf{R_Z}(\rho_P)$, deci matricea de instanţiere a unei planete este $\mathbf{M_P} = \mathbf{R_Z}(\rho_P)\mathbf{T}(r_P,0,0)$.

Transformarea de instanţiere a satelitului se calculează relativ la sistemul de referinţă local al planetei sale. La fel ca şi pentru o planetă, poziţia centrului satelitului depinde de raza de rotaţie (r_S) a acestuia şi unghiul de rotaţie (ρ_S) şi are coordonatele în sistemul de referinţă local al planetei $C_S(r_S \cos\rho_S, r_S \sin\rho_S, 0)$. Rezultă că matricea de instanţiere a satelitului relativ la sistemul de referinţă al planetei sale este $\mathbf{M_S} = \mathbf{R_Z}(\rho_S)\mathbf{T}(r_S,0,0)$. Programul de reprezentare a acestui sistem solar este următorul:

```
#include <gl/glut.h>

#define rp1		1.8
#define rp2		1.2
#define rs1		0.6
static int rop1 = 20, rop2 = 40, ros1 = 20;
static float zpos = 5;

void Init(){
	glClearColor(1.0,1.0,1.0,0.0);
	glEnable(GL_DEPTH_TEST);
}
void Axes(double x,double y, double z){
	// aceeaşi funcţie ca în exemplul precedent
}
void Keyboard(unsigned char key, int x, int y){
	switch (key){
	case 'Z':
		zpos = zpos+0.1;
		glutPostRedisplay();
		break;
	case 'z':
		zpos = zpos-0.1;
		glutPostRedisplay();
		break;
	case 'P':
		rop1 = (rop1+10)%360;
		rop2 = (rop2+10)%360;
		glutPostRedisplay();
		break;
```

```
		case 'p':
			rop1 = (rop1-10)%360;
			rop2 = (rop2-10)%360;
			glutPostRedisplay();
			break;
		case 'S':
			ros1 = (ros1+10)%360;
			glutPostRedisplay();
			break;
		case 's':
			ros1 = (ros1-10)%360;
			glutPostRedisplay();
			break;
		}
}
void Display(void){
		glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

		glPushMatrix();
			glTranslatef(-1,-1,-zpos);		//transf.de obs.
		  Axes(3,1.5,2.5);
		  glColor3f(0.4,0.4,0.4);
			glutSolidSphere(0.6, 16, 16);	// soare

		  glColor3f(0,0,0);
			glPushMatrix();
				glRotatef(rop1, 0,0,1);
				glTranslatef(rp1,0,0);
				glutWireSphere(0.25, 8, 8);	// planeta 1
				Axes(1,0.8,1.8);

				glRotatef(ros1, 0,0,1);
				glTranslatef(rs1,0,0);
				glutSolidSphere(0.1, 8, 8);	// satelit 1
				Axes(0.5,0.5,1);
			glPopMatrix();

			glRotatef(rop2, 0,0,1);
			glTranslatef(rp2,0,0);
			glutWireSphere(0.2, 8, 8);	// planeta 2
			Axes(0.5,1.1,2.5);
		glPopMatrix();
		glutSwapBuffers();
}
void Reshape(int w, int h){
   h = (h == 0) ? 1 : h;
   glViewport(0, 0, w, h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(60.0,(GLfloat)w/(GLfloat)h, 1, 40.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}
int main(int argc, char** argv){
  glutInit(&argc, argv);
		glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB
		    | GLUT_DEPTH );
```

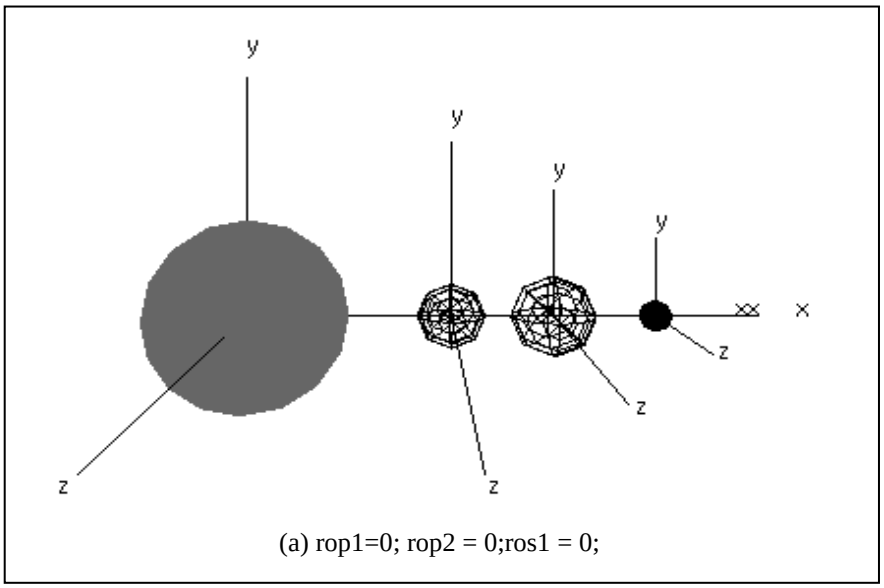
```
        glutInitWindowSize(500,500);
        glutInitWindowPosition(100,100);
        glutCreateWindow("Sistem Solar");
        Init();
        glutDisplayFunc(Display);
        glutReshapeFunc(Reshape);
        glutKeyboardFunc(Keyboard);
        glutMainLoop();
        return 0;
}
```

Imaginile obţinute la execuţia acestui program sunt prezentate în Fig. 4.5. În funcţia `Init()` se validează funcţionarea algoritmului de Z-buffer, pentru ascunderea suprafeţelor ascunse în funcţie de adâncime (coordonata z în sistemul de referinţă normalizat) prin funcţia `glEnable(GL_DEPTH_TEST)`. Valoarea cu care este şters bufferul de adâncime la începutul fiecărui cadru (la execuţia funcţiei `glClear()`) are implicit valoarea maximă a coordonatei z a volumului canonic în sistemul de referinţă normalizat (1.0).

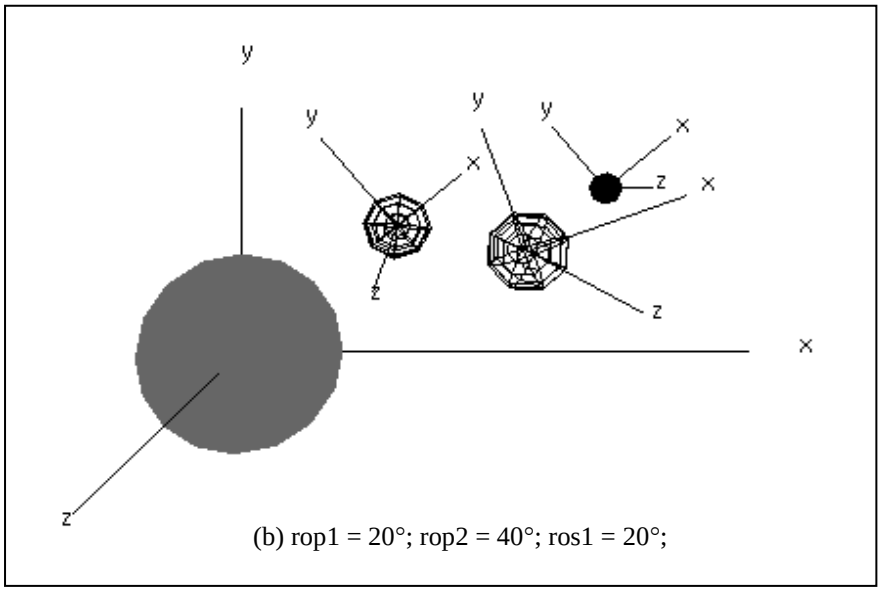
În funcţia `Reshape()` este utilizată o altă funcţie de definire a proiecţiei perspectivă, funcţia `gluPerspective()` din biblioteca GLU, care este o versiune simplificată a funcţiei `glFrustum()`. Prototipul acestei funcţii este:

```
void gluPerspective(GLdouble fovy, GLdouble aspect,
        GLdouble zNear, GLdouble zFar);
```

unde `fovy` este unghiul de vizibilitate pe verticală (în grade); `aspect` este raportul `w/h` a dimensiunilor pe orizontală şi verticală a ferestrei de afişare; `zNear` şi `zFar` sunt valorile absolute ale distanţelor planelor de vizibilitate apropiată, respectiv depărtată.



(a) rop1=0; rop2 = 0;ros1 = 0;



(b) rop1 = 20°; rop2 = 40°; ros1 = 20°;

Fig. 6.5 Două imagini ale sistemului solar,
pentru două poziţii diferite ale planetelor şi satelitului.

Funcţia callback `Keyboard()` este apelată la acţionarea unei taste. În acest exemplu simplu sunt tratate câteva taste. Tasta ‘z’ scade iar tasta ‘Z ‘ creşte valoarea coordonatei z_V a observatorului. Tastele ‘p’ şi ‘P’ scad, respectiv, cresc unghiul de rotaţie al planetelor. Tastele ‘s’ şi ‘S’ scad, respectiv, cresc unghiul de rotaţie al satelitului. Se obţine o mişcare interactivă (foarte simplificată, e adevărat, dar intuitivă) a planetelor şi satelitului.

Funcţia callback `Display()` se execută la fiecare redesenare care are loc la modificarea dimensiunilor ferestrei de afişare şi prin apelul funcţiei `glutPostRedisplay()`, la modificarea poziţiei de observare (`zpos`) sau a uneia dintre valorile unghiurilor `rop1`, `rop2` sau `ros1`.

Păstrarea matricei identitate în stiva de modelare-vizualizare, astfel încât să poată fi folosită pentru o nouă secvenţă de compunere se realizează prin perechea de funcţii `glPushMatrix()` şi `glPopMatrix()` cea mai exterioară din funcţia `Display()`. Matricea de transformare de observare construită în matricea curentă este reţinută în stivă prin cea-a de-a doua funcţie `glPushMatrix()`. Matricea de instanţiere pentru prima planetă necesită introducerea mai întâi a rotaţiei şi apoi a translaţiei şi ea se aplică acesteia. Matricea de instanţiere a satelitului se construieşte prin compunerea matricei sale de instanţiere cu matricea de instanţiere a planetei respective. După instnţiere satelitului, matricea de instanţiere a acestuia nu mai este necesară, funcţia `glPopMatrix()` o elimină din stivă şi vârful stivei conţine acum matricea de transformare de observare, care poate fi folosită pentru compunerea matricei pentru cea de-a doua planetă. În Fig. 4.5 sunt prezentate două imagini captate din fereastra de afişare a programului pentru diferite valori ale poziţiilor planetelor şi

satelitului. Imaginea nefiind color, s-a considerat că reprezentarea wireframe pentru planete este mai sugestivă. Pentru fiecare obiect s-au desenat şi axele de coordonate ale sistemul de referinţă local.

Este interesant de precizat faptul că, în marea majoritate a aplicaţiilor grafice, fiecare imagine (cadru, *frame*) se generează independent de imaginile precedente. Pentru fiecare cadru se calculează poziţia de observare şi poziţia obiectelor în scenă (dacă aceasta variază, aşa cum este situaţia în exemplul sistemului solar) şi imaginea se generează pornind de fiecare dată de la aceste informaţii. Trecerea de la o imagine la următoarea prin acumularea unor valori (de mişcare sau de poziţie) nu aduce nici un beneficiu semnificativ ca viteză de execuţie şi poate produce erori de poziţionare la execuţie un timp îndelungat. Chiar modificarea cu o valoare extrem de mică a poziţiei primitivelor geometrice redate pe display necesită ştergerea imaginii precedente şi construirea uneia noi prin reluarea întregii succesiuni a operaţiilor grafice, deoarece este foarte complicat (practic imposibil) de a se calcula ce părţi ale ecranului rămîn neschimbate şi ce părţi se modifică şi cum se modifică. ν

Decuparea suprafeţelor şi eliminarea obiectelor aflate în exteriorul volumului de vizualizare

În OpenGL decuparea primitivelor geometrice la volumul de vizualizare este efectuată în mod automatizat, fără nici o intervenţie din partea programatorului. Volumul de vizualizare se defineşte o dată cu definirea proiecţiei perspectivă (folosind una din funcţiile `glFrustum()` sau `gluPerspective()`) sau a proiecţiei paralele (folosind funcţia `glOrtho()`). Dacă se urmăreşte diagrama de execuţie din Fig. 6.1, se poate observa că, după ce vârfurile au fost transformate prin înmulţire cu matricea curentă a stivei de modelare-vizualizare, se recompun primitivele geometrice, după care se aplică transformarea de normalizare şi decuparea. În cursul transformării de modelare-vizualizare, vârfurile au fost tratate individual (transformat fiecare în parte), dar pentru decupare este necesar să fie recompuse în primitive, deoarece decuparea se efectuează prin parcurgere în ordine a laturilor poligoanelor.

Fiind o bibliotecă de funcţii de nivel scăzut, care se ocupă de redarea unor primitive geometrice, OpenGL nu prevede nici o facilitate pentru eliminarea obiectelor care sunt în exteriorul volumului de vizualizare (*culling*). Este sarcina programatorului să definească volumele de delimitare, să le transforme din sistemul local în sistemul de referinţă de observare şi să le utilizeze pentru eliminarea obiectelor aflate complet în exteriorul volumului de vizualizare (obiecte "invizibile").

Diferite sisteme de dezvoltare a aplicaţiilor grafice (cum sunt sistemele Sense8 şi Performer) oferă suport pentru definirea şi utilizarea volumelor de delimitare şi eliminare a obiectelor invizibile.

Eliminarea suprafețelor ascunse

În OpenGL este implementat algoritmul Z-buffer de eliminare a suprafeţelor ascunse. Numărul de biţi/pixeli ai bufferului de adâncime se stabileşte la definirea formatului pixelului. Implicit, se setează bufferul de adâncime cu 32 biţi/pixel. Dacă

se foloseşte utilitarul GLUT, atunci funcţia de iniţializare a modului de afişare trebuie să conţină opţiunea GLUT_DEPTH:

```
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH)
```

La iniţializarea bibliotecii OpenGL testul de adâncime este invalidat. Pentru validarea lui se apelează funcţia glEnable(GL_DEPTH_TEST), iar la începutul fiecărui cadru de imagine trebuie să fie şters şi bufferul de adîncime, o dată cu ştergerea bufferului de imagine, ceea ce se obţine prin introducerea opţiunii GL_DEPTH_BUFFER_BIT în apelul funcţiei glClear():

```
glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Valoarea cu care se "şterge " bufferul de adâncime este implicit valoarea maximă 1, iar testul care se efectuează implicit este de înscriere a noului pixel dacă are adâncimea mai mică decât valoarea din buffer (GL_LESS). În OpenGL mai există posibilitatea de a defini altă valoare de ştergere a bufferului de adâncime (prin funcţia glClearDepth(GLclampd depth)), ca şi stabilirea testului de comparaţie, prin funcţia

```
void glDepthFunc(GLenum func);
```

unde func poate fi una din mai multe constante simbolice admise (GL_NEVER, GL_LESS, GL_EQUAL, GL_LEQUAL, GL_GREATER, GL_NOTEQUAL, GL_GEQUAL, GL_ALWAYS).

Selecția suprafețelor în funcție de orientare

OpenGL oferă suport direct pentru selecţia primitivelor geometrice în funcţie de orientarea lor. O primitivă geometrică este de categoria cu orientare directă (spre faţă - *frontface*) dacă sensul de parcurgere al acesteia (ordinea vârfurilor date prin funcţiile glVertex#() cuprinse în blocul glBegin(),glEnd()), este aceeaşi cu ordinea stabilită prin funcţia glFrontFace() (în sensul acelor de ceas sau în sens invers acelor de ceas). Alftel, primitiva are orientare inversă (spre spate - *backface*). Prin funcţia glCullFace(GLenum type) se selectează care tip de suprafeţe trebuie să fie eliminate: dacă argumentul type are valoare GL_BACK sunt eliminate primitivele cu orientare inversă; dacă are valoarea GL_FRONT sunt eliminate primitivele cu orientare directă. Testul de orientare şi eliminarea primitivelor se execută numai dacă a fost validat acest test, prin apelul funcţiei glEnable(GL_CULL_FACE).

În segmentul de cod care urmează, cele două poligoane sunt cu orientare directă (front) deoarece vârfurile lor sunt date în sensul invers acelor de ceas, aşa cum este setarea implicită a sensului de parcurgere directă în OpenGL. Primul poligon este desenat, deoarece este activă opţiunea de eliminare a primitivelor cu orientare inversă, iar al doilea nu este desenat, deoarece este activă opţiunea de eliminare a primitivelor cu orientare directă.

```
void Display(){
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
        glEnable(GL_CULL_FACE);// validează selectia
```

```
        glCullFace(GL_BACK);       // se elimina primitive inverse
        glColor3f(0.5,0.5,0.5);
        glBegin(GL_POLYGON);    //orientat direct, se deseneaza
                glVertex3d(3,3,-10);
                glVertex3d(4,3,-10);
                glVertex3d(4,4,-10);
                glVertex3d(3,4,-10);
        glEnd();
        glCullFace(GL_FRONT);     // se elimina primitive directe
        glBegin(GL_POLYGON);//orientat direct, nu se deseneaza
                glVertex3d(1,1,-10);
                glVertex3d(2,1,-10);
                glVertex3d(2,2,-10);
                glVertex3d(1,2,-10);
        glEnd();
}
```

Liste de display OpenGL

O listă de display este un grup de comenzi OpenGL care sunt memorate pentru a fi utilizate ulterior. Atunci când este apelată o listă de display, comenzile pe care le conţine sunt executate în ordinea în care ele au fost memorate. Listele de display pot îmbunătăţi performanţele de execuţie atunci când se redesenează aceleaşi forme geometrice de mai multe ori.

Definirea unei liste de display începe cu apelul funcţiei `glNewList()` şi se termină la apelul funcţiei `glEndList()`. Prototipul funcţia `glNewList` este:

```
void glNewList(GLuint list, GLenum mode);
```

Parametrul `list` este un indice care identifică lista de display şi este folosit la apelul acesteia. Parametrul `mode` poate lua una din constantele simbolice `GL_COMPILE` sau `GL_COMPILE_AND_EXECUTE`. Modul `GL_COMPILE` are ca efect compunerea listei de displaz. Modul `GL_COMPILE_END_EXECUTE` are ca efect compunerea listei şi execuţia ei imediată.

O listă de display cu indicele `list` este executată la apelul funcţiei:

```
void glCallList(GLuint list);
```

Ca exemplu de utilizare a unei liste de display se reiau comenzile de desenare a unui octogon din Exemplul 6.2.

```
#include <math.h>
#include <gl\glut.h>
#define PI 3.141592
void DispList1(){
        glNewList(1,GL_COMPILE);
        int n = 8;
        double radius = 10;
        glColor3d(0,0,0);
        glBegin(GL_LINE_LOOP);
                for (int i=0;i<n;i++){
                        double angle = 2*PI*i/n;
                        glVertex3d(radius*cos(angle),
```

```
                    radius*sin(angle),-40);
                    }
            glEnd();
            glEndList();
}
 void Init(){
            glClearColor(1.0,1.0,1.0,1.0);
            DispList1();
}
 void  Display(){
   glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
             glCallList(1);
   glutSwapBuffers();
}
```

Funcţiile `Reshape()` şi `main()` ale programului sunt cele prezentate în programele precedente. Funcţia `DisplayList()` apelată la iniţializare crează lista de display cu indicele 1 care conţine calculul coordonatelor octogonului şi OpenGL de desenare a acestuia. În funcţia `Display()` este apelată funcţia `glCallList(1)`, care redesenează octogonul la fiecare apel.

Avantajul listelor de display este semnificativ atunci când, pentru generarea unei imagini, se folosesc funcţii de calcul combinate cu comenzi OpenGL: calculele se execută o singură dată, la crearea listei, iar comenzile OpenGL rezultate se execută la fiecare apel al listei.

O astfel de situaţie este prezentă şi în exemplul de mai sus şi apare frecvent în operaţiile de iluminare, umbrire şi texturare a obiectelor.

Texturarea în OpenGL

Biblioteca OpenGL permite texturarea obiectelor, oferind suport pentru un număr foarte mare de posibilităţi de definire, aplicare şi filtrare a texturilor. În lucrare vor fi prezentate numai o parte dintre aceste posibilităţi, ca exemplificare a modului de implementare a texturării şi ca bază de pornire în crearea aplicaţiilor grafice. Celelate aspecte, mai de detaliu, se pot urmări în manualul de referinţă al bibliotecii, precum şi în numeroasele exemple care sunt disponibile în OpenGL şi GLUT. .

Se pot defini texturi unidimensionale sau bidimensionale, cu rezoluţii (număr de texeli) care se specifică la crearea imaginii texturii. Fiecare texel constă din una, două, trei sau patru componente, reprezentând valori de modulare sau cvadruple RGBA. Texturarea poate fi validată sau invalidată prin apelul funcţiei `glEnable()`, respectiv `glDisable()`, cu argument una din constantele simbolice `GLTEXTURE_1D` sau `GL_TEXTURE_2D`, pentru texturi unidimensionale sau bidimensionale.

Definirea texturilor

În OpenGL se pot defini mai multe texturi (în general, la iniţializarea programului) şi fiecare suprafaţă se texturează folosind una dintre acestea. Fiecare textură (obiect textură - *texture object*) are un nume unic în program şi este definită printr-un set de date care permit aplicarea acesteia suprafeţelor: adresa imaginii

texturii, funcţia texturii (modulare, înlocuire, combinare), şi proprietăţile texturii (filtrări, mod de repetare, etc).

Posibilitatea de a manevra mai multe texturi în timpul execuţiei, fără să fie necesar încărcarea sau generarea imaginii texturii de fiecare dată, îmbunătăţeşte performanţele de texturare. Deoarece OpenGL este o interfaţă de programare scrisă în limbajul C, obiectele textură sunt definite prin date şi funcţii separate, care trebuie să fie specificate într-o anumită ordine:

(a) Generarea numelor texturilor.
(b) Crearea obiectelor textură şi conectarea lor (*bind*) la numele texturilor.
(c) Activarea unei texturi, pentru aplicarea acesteia primitivelor geometrice care urmează.

Numele texturii. Numele texturilor sunt numere întregi fără semn (de tipul `GLuint`) care sunt memorate într-un vector transmis ca argument funcţiei:

```
void glGenTextures(GLsizei n, GLuint *textureNames);
```

Această funcţie creează un vector de `n` nume de texturi, unice în program, pe care le memorează în vectorul `textureNames`. Numele create nu sunt neapărat numere succesive. Funcţia `glGenTextures()` creează numai numele texturilor şi le marchează ca utilizate, dar obiectele textură se creează numai la conectarea (*bind*) acestora.

Crearea texturilor. Funcţia `glBindTexture()` se foloseşte atât pentru crearea cât şi pentru utilizarea unei texturi. Prototipul ei este:

```
void glBindTexture(GLenum target, GLuint texName);
```

Parametrul `target` poate avea ca valoare una din constantele simbolice `GL_TEXTURE_1D` sau `GL_TEXTURE_2D`, pentru texturi unidimensionale, respectiv bidimensionale. Argumentul `texName` este numele unei texturi, generat de funcţia `glGenTextures()` şi memorat în vectorul de nume ale texturilor.

Atunci când este apelată prima oară pentru un nume de textură, funcţia `glBindTexture()` creează un nou obiect textură, cu toate datele referitoare la imaginea şi proprietăţile texturii implicite. După apelul funcţiei `glBindTexture()`, textura cu numele dat ca argument devine textură curentă şi toate operaţiile ulterioare, atât pentru definirea unor proprietăţi ale texturii, cât şi pentru aplicaţia texturii, folosesc textura curentă. Textura curentă se schimbă prin apelul unei noi funcţii `glBindTexture()`.

Crearea imaginii de textură. Imaginea texturii este un tablou unidimensional sau bidimensional de texeli, fiecare texel având una, două, trei sau patru componente. Semnificaţia componentelor texelilor se stabileşte la crearea imaginii texturii prin definirea formatului intern al imaginii printr-un argument al uneia din funcţiile `glTexImage1D()`, `glTexImage2D()`.

Texturile unidimensionale au o utilizare restrânsă. Ele sunt folosite pentru texturarea în benzi, pentru care variaţia culorii are o singură direcţie. În continuare se

vor prezenta texturile bidimensionale, care sunt cel mai frecvent folosite. Funcţia `glTexImage2D()` are următorul prototip:

```
void glTexImage2D(GLenum target,GLint level,
     GLint internalFormat,GLsizei width,
     GLsizei height,GLint border,
     GLenum format,GLenum type,const GLvoid *pixels);
```

Parametrul `target` specifică crearea unei imagini de textură bidimensională (prin constanta simbolică `GL_TEXTURE_2D`) sau o interogare a capacităţii de memorare a unei imagini de textură (prin constanta simbolică `GL_PROXY_TEXTURE_2D`).

Parametrul `level` indică nivelul imaginii într-o succesiune de imagini prefiltrate de tipul mip-map. Nivelul 0 reprezintă imaginea originală, cu rezoluţia maximă. Celelate nivele se pot crea automat din imaginea originală (prin funcţia `gluBuild2DMipmaps()`) sau pot fi create din tablouri de texeli, cu acelaşi format ca imaginea de nivel 0 a texturii.

Parametrul `internalFormat` indică tipul componentelor texelilor texturii, ca intensitate, luminanţă sau R, G, B, A. Valoarea argumentului `internalFormat` poate fi un întreg între 1 şi 4 sau una dintre treizeci şi două de constante simbolice, dintre care cele mai frecvent folosite sunt: `GL_LUMINANCE`, `GL_LUMINANCE_ALPHA`, `GL_RGB`, `GL_RGBA`. Aceste constante corespund valorilor 1, 2, 3, 4 ale argumentului. Aceste componente ale texelilor sunt combinate cu componentele culorii proprii (înainte de texturare) a pixelului pentru a crea culoarea finală a pixelului. Modul de combinare depinde de funcţia de texturare, descrisă imediat mai jos.

Parametrii `width` şi `height` dau dimensiunea imaginii texturii, în număr de texeli, pe orizontală şi pe verticală.

Parametrul `border` reprezintă lăţimea borderului imaginii de textură. Această caracteristică permite alipirea mai multor porţiuni de imagini într-o singură imagine de textură, iar borderul este folosit în operaţiile de filtrare, în care sunt necesari texeli vecini texelului curent, pentru medierea culorilor acestora. Devizarea unei imagini de textură în mai multe părţi este impusă în anumite implementări ale bibliotecii OpenGL, atunci când texturarea este implementată hardware şi imaginea de textură se încarcă într-o memorie rapidă din acceleratorul grafic. Dacă imaginea texturii este prea mare pentru a fi stocată în întregime în memoria de textură, atunci se poate împărţi în mai multe părţi, care sunt încărcate şi folosite succesiv. În această situaţie borderul permite filtrarea corectă la graniţele părţilor componente ale imaginii.

Parametrii `format` şi `type` se referă la formatul şi tipul datelor imaginii de textură. Parametrul format poate fi una din constantele simbolice `GL_COLOR_INDEX`, `GL_RGB`, `GL_RGBA`, `GL_RED`, `GL_GREEN`, `GL_BLUE`, `GL_ALPHA`, `GL_LUMINANCE`, `GL_LUMINANCE_ALPHA`. Parametrul `type` poate fi `GL_BYTE`, `GL_UNSIGNED_BYTE`, `GL_SHORT`, `GL_UNSIGNED_SHORT`, `GL_INT`, `GL_UNSIGNED_INT`, `GL_FLOAT sau GL_BITMAP`.

Ultimul parametru, `pixels`, conţine datele imaginii de texturii.

La apelul funcţiei `glTexImage2D()`, se creează imaginea texturii curente (cea conectată prin funcţia `glBindTexture()`), în formatul specificat.

Funcţii de texturare. Modul în care se calculează culoarea unui pixel (sau fragment de pixel) se stabileşte prin apelul uneia din funcţiile `glTexEnv#()`:

```
void glTexEnvf(GLenum target,GLenum pname,GLfloat param);
void glTexEnvi(GLenum target,GLenum pname,GLfloat param);
```

unde argumentele au următoarea semnificaţie:

- `target` specifică mediul (*environment*) de aplicare a texturii; trebuie să fie constanta simbolică `GL_TEXTURE_ENV`.
- `pname` este numele simbolic al unui paramentru de mediu de texturare; trebuie să aibă valoarea `GL_TEXTURE_ENV_MODE`.
- `param` specifică funcţia de texturare printr-o constantă simbolică care poate lua una din valorile `GL_MODULATE`, `GL_DECAL`, `or` `GL_BLEND`.

Funcţiile de texturare stabilesc modul de calcul al culorii rezultate a pixelilor pe baza culorii texelilor din imaginea de textură şi a culorii pixelilor primitivei geometrice (care se obţin din culoarea curectă sau din calcule de umbrire). În general, în modul `GL_DECAL`, culoarea finală atribuită unui pixel este culoarea texturii; în modul `GL_MODULATE`, se foloseşte culoarea texturii pentru modularea culorii fragmentului (pixelului); în modul `GL_BLEND` se combină culoarea texturii cu culoarea fragmentului. Această modalitate generală de atribuire a culorii depinde şi de formatul intern al texturii, care se stabileşte la crearea imaginii de textură.

Atribuirea coordonatelor de texturare. Coordonatele vârfurilor primitivelor geometrice în planul texturii (coordonatele de texturare) se transmit prin funcţiile `glTexCoord#()`, care pot primi 1, 2, 3 sau 4 argumente de diferite tipuri, rezultând un număr mare de posibilităţi de apel. O parte dintre acestea sunt:

```
void glTexCoord1f(GLfloat s);
void glTexCoord2f(GLfloat s,GLfloat t);
void glTexCoord3f(GLfloat s,GLfloat t,GLfloat r);
void glTexCoord4f(GLfloat s,GLfloat t,GLfloat r,
                                        GLfloat q);
void glTexCoord1f(GLdouble s);
void glTexCoord2f(GLdouble s,GLdouble t);
void glTexCoord3f(GLdouble s,GLdouble t,GLdouble r);
void glTexCoord4f(GLdouble s,GLdouble t,GLdouble r,
                                        GLdouble q);
```

Coordonatele de texturare pot avea 1, 2 sau 3 pentru texturi unidimensionale, bidimensionale sau tridimensionale. În spaţiul texturii coordonatele sunt notate `s`, `t`, `r`, corspunzător coordonatelor `x`, `y`, `z` în spaţiul obiect. Cea de-a patra componentă, `q`, este componenta de scală în reprezentarea texturii într-un sistem de coordonate omogen, asemănător sistemului de coordonate omogen folosit pentru reprezentarea

punctelor în spaţiu. Această coordonată este folosită dacă sunt necesare transformări ale texturi în coordonate omogene. În versiunea folosită în momentul de faţă a bibliotecii OpenGL (Versiunea 1.1), nu este încă implementată texturarea spaţială şi coordonata `r` nu este folosită, fiind prevăzută ca rezervă pentru utilizare ulterioară.

Dacă valorile coordonatelor de texturare în vârfurile primitivelor grafice sunt cuprinse în intervalul [0,1], atunci textura este aplicată o singură dată pe suprafaţa respectivă. Dacă aceste valori depăşesc intervalul [0,1], atunci textura poate fi repetată pe suprafaţă sau limitată la intervalul [0,1]. Proprietatea unei texturi de a fi repetată sau limitată se stabileşte prin apelul uneia din funcţiile:

```
void glTexParameterf(GLenum target,GLenum pname,
                GLfloat param);
void glTexParameteri(GLenum target,GLenum pname,
                                                GLint param);
```

În aceste funcţii, parametrul `target` reprezintă tipul texturii şi poate lua una din constantele simbolice `GL_TEXTURE_1D` sau `GL_TEXTURE_2D`. Parametrul `pname` specifică numele simbolic al unei proprietăţi a texturii. şi poate lua una din constantele: `GL_TEXTURE_MIN_FILTER`, `GL_TEXTURE_MAX_FILTER`, `GL_TEXTURE_WRAP_S`, `GL_TEXTURE_WRAP_T`. Primele două valori se referă la opţiunile de filtrare ale texturii şi vor fi prezentate în paragraful următor. Următoarele valori, setează proprietatea de repetare a texturii pentru coordonata `s`, respectiv `t`. În acest caz, parametrul `param` poate fi `GL_REPEAT`, pentru repetarea texturii, sau `GL_CLAMP`, pentru limitarea texturii la intervalul [0,1].

Mai multe aspecte privind definirea şi folosirea texturilor în OpenGL vor fi detaliate în exemplele care urmează.

ν **Exemplul 6.5**

În acest exemplu este prezentat programul prin care se aplică texturi în tablă de şah unor suprafeţe în spaţiu. Imaginea pe care o generează este o variantă a celei din Fig. 5.15.

```
#include <GL/glut.h>

#define imageWidth 64
#define imageHeight 64

static GLubyte image4[imageHeight][imageWidth][4];
static GLubyte image8[imageHeight][imageWidth][4];
static GLuint texName[2];

void Makeimages(){
        int c;
        for (int i=0;i<imageHeight;i++){
                for (int j=0;j<imageWidth;j++){
                        c = (((i&0x4)==0)^((j&0x4)==0)) *255;
                        image4[i][j][0] = (GLubyte) c;
                        image4[i][j][1] = (GLubyte) c;
                        image4[i][j][2] = (GLubyte) c;
                        image4[i][j][3] = (GLubyte) 255;
```

```
                                    c = (((i&0x8)==0)^((j&0x8)==0)) *255;
                                    image8[i][j][0] = (GLubyte) c;
                                    image8[i][j][1] = (GLubyte) c;
                                    image8[i][j][2] = (GLubyte) c;
                                    image8[i][j][3] = (GLubyte) 255;
                            }
                    }
}
void Init(void){
            glClearColor(0.6, 0.6, 0.6, 1.0);
  glEnable(GL_DEPTH_TEST);
            glShadeModel(GL_FLAT);

            /* TEXTURARE */
            Makeimages();
      glPixelStorei(GL_UNPACK_ALIGNMENT,1);
      glGenTextures(2, texName); // se creaza doua texturi

            /* Definirea primei texturi */
            glBindTexture(GL_TEXTURE_2D,texName[0]);
            glTexImage2D(GL_TEXTURE_2D,0,GL_RGBA,imageWidth,
        imageHeight,0,GL_RGBA,GL_UNSIGNED_BYTE,image4);
            glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
          GL_DECAL);
            glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,
              GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,
              GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,
              GL_NEAREST);
            glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,
              GL_NEAREST);

            /* Definirea celui de-a doua texturi */
            glBindTexture(GL_TEXTURE_2D, texName[1]);
            glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, imageWidth,
     imageHeight, 0, GL_RGBA, GL_UNSIGNED_BYTE,image8);
            glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
          GL_DECAL);
            glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,
              GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,
              GL_REPEAT);
            glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,
              GL_NEAREST);
            glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,
              GL_NEAREST);

            glEnable(GL_TEXTURE_2D);
}
void Display(){
            glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);

            glBindTexture(GL_TEXTURE_2D, texName[1]);

            glPushMatrix();
            glTranslated(-2.0,0.0,-8);
```

```
            glBegin(GL_QUADS);
                        glTexCoord2f(0.0,0.0);glVertex3f(-0.5,-1.0,0.0);
                        glTexCoord2f(0.5,0.0);glVertex3f( 0.5,-1.0,0.0);
                        glTexCoord2f(0.5,1.0);glVertex3f( 0.5, 1.0,0.0);
                        glTexCoord2f(0.0,1.0);glVertex3f(-0.5, 1.0,0.0);
            glEnd();
            glPopMatrix();

            glPushMatrix();
            glTranslated(0.0,0.0,-8);
            glRotated(-70,1.0,0.0,0.0);
            glBegin(GL_QUADS);
                        glTexCoord2f(0.0,0.0);glVertex3f(-0.5,-1.0,0.0);
                        glTexCoord2f(0.5,0.0);glVertex3f( 0.5,-1.0,0.0);
                        glTexCoord2f(0.5,1.0);glVertex3f( 0.5, 1.0,0.0);
                        glTexCoord2f(0.0,1.0);glVertex3f(-0.5, 1.0,0.0);
            glEnd();
            glPopMatrix();

            glBindTexture(GL_TEXTURE_2D, texName[0]);
            glPushMatrix();
            glTranslated(1.5,0.0,-8);
            glBegin(GL_QUADS);
                        glTexCoord2f(0.5,0.0);glVertex3f(0.0,-1.0,0.0);
                        glTexCoord2f(1.0,0.0);glVertex3f( 1.0,-1.0,0.0);
                        glTexCoord2f(1.0,1.0);glVertex3f( 1.0, 1.0,0.0);
                        glTexCoord2f(0.5,0.5);glVertex3f(-0.0, 0.0,0.0);
            glEnd();
            glPopMatrix();

            glutSwapBuffers();
}
void Reshape(int w, int h){
   h = (h == 0) ? 1 : h;
   glViewport(0, 0, w, h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(60.0, (GLfloat) w/(GLfloat) h,
               0.1, 4000.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}
int main(int argc, char** argv){
  glutInit(&argc, argv);
            glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
            glutInitWindowSize(500,500);
            glutInitWindowPosition(100,100);
            glutCreateWindow("Texture Check Image");
            Init();
            glutDisplayFunc(Display);
            glutReshapeFunc(Reshape);
            glutMainLoop();
            return 0;
}
```

Imaginea captată din fereastra afişată la execuţia acestui program este dată în Fig. 6.6 şi diferă de imaginea din Fig. 5.15 prin faptul că se folosesc două texturi.

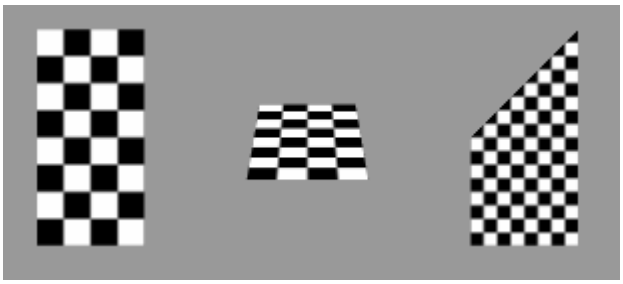


Fig. 6.6 Aplicaţia texturilor pe suprafeţe plane

În mod obişnuit, imaginile texturilor se citesc din fişiere, dar texturile simple, în tablă de şah, se pot genera prin program. Funcţia `Makeimages()` creează două tablouri de 64 x 64 locaţii, fiecare locaţie fiind un vector cu patru componente R, G, B, A. În primul tablou (image4) este creată o tablă cu 16 x 16 pătrate alternante de culoare albă şi neagră, fiecare pătrat de dimensiune 4 x 4 texeli. În al doilea tablou (image8) este creată o tablă cu 8 x 8 pătrate alternante de culoare albă şi neagră, fiecare pătrat de dimensiune 8 x 8 texeli.

Texturile sunt definite în funcţia `Init()`. Mai întâi se creează numele a două texturi în vectorul `texName[2]`, prin funcţia `glGenTextures()`. Pentru crearea şi definirea proprietăţilor fiecărei texturi, se conectează mai întâi textura specificată prin numele ei (funcţia `glBindTexture()`) şi apoi se specifică proprietăţile texturii.

În funcţia `Display()` se utilizează texturile definite pentru texturarea unor primitive geometrice. Textura care se aplică este textura curentă, activată prin numele ei dat ca argument funcţiei de conectare `glBindTexture()`. Ca urmare, primele două suprafeţe sunt texturate cu textura cu numele `texName[1]`, iar a treia suprafaţă este texturată cu textura cu numele `texName[0]`, ceea ce se poate observa în imaginea din Fig. 4.15. Imaginea din Fig. 3.65 a fost generată cu acelaţi program, dar s-a activat numai textura cu numele `texName[1]`, adică textura formată din 8 x 8 pătrate. Într-un bloc `glBegin()- glEnd()` se transmit vârfurile unei primitive geometrice; pentru fiecare vârf se definesc mai întâi coordonatele de texturare (cu `glTexCoord2f()`) şi apoi coordonatele spaţiale ale vârfului (cu `glVertex3f()`). Restul funcţiilor din program efectuează operaţiile deja cunoscute, de transformări geometrice, setare culori de ştergere, etc. ν

Filtrarea texturilor

Tipul de filtrare care se aplică unei texturi este definit prin valorile a doi parametri, `GL_TEXTURE_MAG_FILTER şi GL_TEXTURE_MIN_FILTER`. setaţi prin apelul funcţiei `glTexParameter#()`. Filtrarea de mărire (*magnification*, `GL_TEXTURE_MAG_FILTER`) se aplică atunci când dimensiunea pre-imaginii pixelului este egală sau mai mică decât dimensiunea texelului (Fig. 6.7(a)). Filtrarea

de micşorare (*minification*, `GL_TEXTURE_MIN_FILTER`) se aplică atunci când dimensiunea pre-imaginii pixelului este mai mare decât dimensiunea texelului (Fig. 6.7(b)).

Dacă nu este definită imagine mip-map a texturii, cei doi parametri de filtrare `GL_TEXTURE_MAG_FILTER` şi `GL_TEXTURE_MAG_FILTER` pot lua numai una din valorile `GL_NEAREST` sau `GL_LINEAR`. Valoarea `GL_NEAREST` înseamnă, de fapt, lipsa filtrării: se selectează texelul cel mai apropiat de centrul pre-imaginii pixelului. Valoarea `GL_LINEAR` asigură filtrarea texturii prin calculul mediei ponderate a patru texeli cei mai apropiaţi de centrul pre-imaginii pixelului. Dacă este implementată software, filtrarea texturii (`GL_LINEAR`) este executată mai lent decât eşantionarea acesteia (`GL_NEAREST`).

În unele cazuri nu este simplu de decis dacă trebuie să se execute o filtrare de mărire sau o filtrare de micşorare. Patrulaterul prin care se reprezintă pre-imaginea pixelului poate să aibă dimensiunea într-o direcţie mai mică decât dimensiunea texelului, iar în altă direcţie mai mare decât dimensiunea texelului. În astfel de situaţii OpenGL selectează filtrul care dă cel mai bun rezultat posibil.

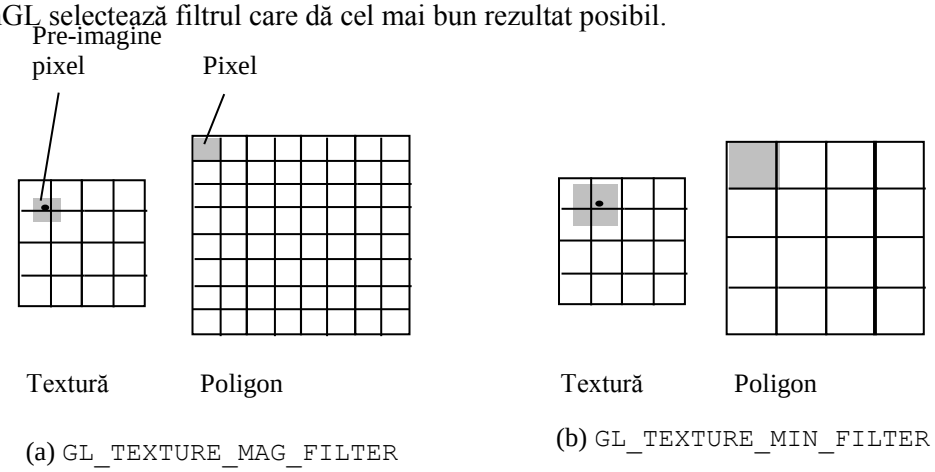


Fig. 6.7 (a) Filtrarea de mărire a texturii.
(b)filtrarea de micşorare a texturii.

Filtrarea de micşorare este accelerată prin tehnica de prefiltrare a texturilor folosind imagini de texturi mip-map. Secvenţa de imagini ale texturii mip-map poate fi generată prin apelul funcţiei `glTexImage2D()` pentru fiecare nivel, începând cu nivelul 0 (rezoluţie maximă) până la ultimul nivel, cu rezoluţie 1 x 1. Datele fiecărei imagini (tabloul bidimensional de texeli) se obţin din imaginea precedentă prin înlocuirea fiecărui grup de patru texeli cu un texel a cărui culoare este media culorilor celor patru texeli din imaginea precedentă.

Dacă s-a definit imaginea mip-map a texturii, atunci filtrarea de micşorare se poate realiza în mai multe moduri, depinzând de felul în care se selectează nivelul imaginii mip-map şi de tipul de filtrare în imaginea mip-map selectată. Filtrul de micşorare (`GL_TEXTURE_MIN_FILTER`) este definit printr-una din următoarele constante simbolice:

- `GL_NEAREST`: nu se aplică nici-o filtrare, se selectează texelul cel mai apropiat de centrul pre-imaginii pixelului din imaginea de nivel 0 a texturii mip-map.
- `GL_LINEAR`: se selectează imaginea de nivel 0 a texturii mip-map şi se mediază ponderat patru texeli cei mai apropiaţi de centrul pre-imaginii pixelului.
- `GL_NEAREST_MIPMAP_NEAREST`: se selectează nivelul imaginii mip-map a texturii pentru care dimensiunea pre-imaginii pixelului este cea mai apropiată de dimensiunea texelului şi în această imagine se selectează texelul cel mai apropiat de centrul pre-imaginii pixelului.
- `GL_LINEAR_MIPMAP_NEAREST`: se selectează nivelul imaginii mip-map a texturii pentru care dimensiunea pre-imaginii pixelului este cea mai apropiată de dimensiunea texelului şi în această imagine se mediază ponderat patru texeli cei mai apropiaţi de centrul pre-imaginii pixelului.
- `GL_NEAREST_MIPMAP_LINEAR`: se selectează două imagini mip-map în care pre-imaginea pixelului are dimensiunea cea mai apropiată de imaginea texelului; în fiecare din aceste imagini se selectează câte un texel după criteriul `GL_NEAREST` (texelul cel mai apropiat de centrul pre-imaginii pixelului); aceşti texeli se mediază ponderat pentru obţinerea valorii finale a culorii pixelului.
- `GL_LINEAR_MIPMAP_LINEAR`: se selectează două imagini mip-map în care pre-imaginea pixelului are dimensiunea cea mai apropiată de imaginea texelului; în fiecare din aceste imagini se selectează câte un texel după criteriul `GL_LINEAR` (media ponderată a patru texeli cei mai apropiaţi de centrul pre-imaginii pixelului) şi aceste valori se mediază ponderat pentru obţinerea valorii finale a culorii pixelului.

Filtrarea `GL_LINEAR_MIPMAP_LINEAR`, care se mai numeşte şi filtrare triliniară, este cel mai eficient mod de eliminare a zgomotului de aliasing al texturii, dar şi cel mai costisitor din punct de vedere al puterii de calcul. Chiar din simpla enumerare a operaţiilor efectuate, se poate observa cantitatea de calcule extrem de mare necesară pentru fiecare pixel al imaginii generate. Filtrarea trilineară a texturilor generează imagini deosebit de realiste, dar este fie lentă, dacă este executată software, fie costisitoare, dacă este implementată hardware.

În exemplul următor se prezintă modul de programare a filtrării texturilor şi rezultatele care se pot obţine cu diferite tipuri de filtrări.

ν **Exemplul 6.6**

```
#include <GL/glut.h>
#include <gl/glaux.h>
AUX_RGBImageRec *image;

void Cube(){
        glBegin(GL_QUADS);
                glTexCoord2f(0.0,0.0);glVertex3f(-1.0,-1.0,1.0);
                glTexCoord2f(1.0,0.0);glVertex3f( 1.0,-1.0,1.0);
```

```
                        glTexCoord2f(1.0,1.0);glVertex3f( 1.0, 1.0,1.0);
                        glTexCoord2f(0.0,1.0);glVertex3f(-1.0, 1.0,1.0);

                        glTexCoord2f(0.0,0.0);glVertex3f(-1.0, 1.0, 1.0);
                        glTexCoord2f(1.0,0.0);glVertex3f( 1.0, 1.0, 1.0);
                        glTexCoord2f(1.0,1.0);glVertex3f( 1.0, 1.0,-2.0);
                        glTexCoord2f(0.0,1.0);glVertex3f(-1.0, 1.0,-2.0);

                        glTexCoord2f(0.0,0.0);glVertex3f(1.0,-1.0, 1.0);
                        glTexCoord2f(1.0,0.0);glVertex3f(1.0,-1.0,-2.0);
                        glTexCoord2f(1.0,1.0);glVertex3f(1.0, 1.0,-2.0);
                        glTexCoord2f(0.0,1.0);glVertex3f(1.0, 1.0, 1.0);
                        ……..…
            glEnd();
}
void Init(void){
   glClearColor(1.0, 1.0, 1.0, 1.0);
   glEnable(GL_CULL_FACE);
            glEnable(GL_DEPTH_TEST);
   image = auxRGBImageLoad("../textures/brick.rgb");
            //image = auxRGBImageLoad("../textures/circles.rgb");
   glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
   gluBuild2DMipmaps(GL_TEXTURE_2D, 3, image->sizeX,
     image->sizeY,GL_RGB,GL_UNSIGNED_BYTE,image->data);
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
          GL_DECAL);
            glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,
              GL_REPEAT);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,
              GL_REPEAT);
   glEnable(GL_TEXTURE_2D);
}
void Display(){
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
            // Primul obiect,fara filtrarea texturii
   glPushMatrix();
            glTranslatef(-2.0, -2.5, -8.0);
            glRotated(-30, 0.0,1.0,0.0);
   glRotated( 20, 1.0,0.0,0.0);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                                 GL_NEAREST);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                                 GL_NEAREST);
            Cube();
   glPopMatrix();
            // Al doilea obiect,cu filtrarea texturii
   glPushMatrix();
            glTranslatef(-2.0, -2.5, -8.0);
            glRotated(20, 0.0,1.0,0.0);
   glRotated(20, 1.0,0.0,0.0);
            glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                                 GL_LINEAR);
   glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                        GL_LINEAR_MIPMAP_LINEAR);
            Cube();
   glPopMatrix();
   glutSwapBuffers();
```

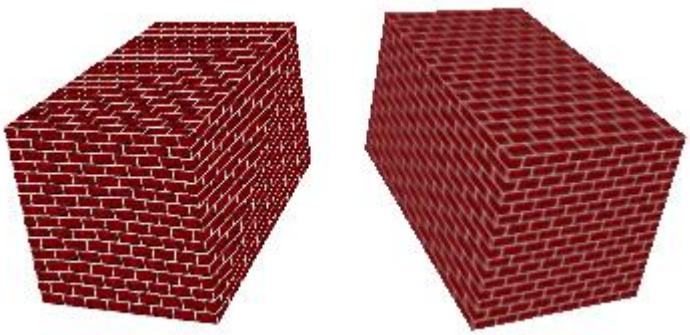
```
}
void Reshape(int w, int h){
   h = (h == 0) ? 1 : h;
   glViewport(0, 0, w, h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(60.0,(GLfloat)w /(GLfloat)h,0.1,1000);
            glMatrixMode(GL_TEXTURE);
   glLoadIdentity();
            glScaled(4,4,4);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}
```

Funcţia `main()` este identică cu cea din exemplul anterior şi nu a mai fost prezentată din nou. În funcţia `Reshape()` se încarcă matricea din vârful stivei matricelor de texturare cu o matrice de scalare, care multiplică coordonatele de texturare cu factorul 4.

În acest program se foloseşte o singură textură, care este în permanenţă textura curentă. Toate funcţiile de definire a parametrilor şi de aplicare a texturii se referă la acestă unică textură şi nu a mai fost necesar atribuirea unui nume şi conectarea (*bind*) texturii pentru definire şi activare.

Imaginea texturii se citeşte dintr-un fişier, folosind funcţia `auxRGBImageLoad()` care aparţine unei biblioteci auxiliare OpenGL, `glaux.lib`, care nu a fost încă descrisă. Acestă bibliotecă auxiliară este o versiune mai veche a sistemului de dezvoltare `GLUT`, care a înlocuit-o pentru cea mai mare parte din funcţiile de creare a ferestrei de afişare OpenGL, tratare a evenimentelor din sistem sau generarea unor obiecte tridimensionale. Deoarece în versiunea 3.6 a sistemului `GLUT` în care au fost scrise programele, nu există o funcţie de încărcare a imaginilor de textură, s-a apelat funcţia din `glaux.lib`. Imaginile în format. Prin funcţii ale bibliotecii `glaux.lib` se pot încărca imagini de textură în format RGB sau în format bitmap DIB (*Device Independent Bitmap*).

Imaginea citită din fişier este folosită pentru crearea secvenţei mip-map de imagini prefiltrate ale texturii. Acestă operaţie se poate efectua prin funcţia `gluBuild2DMipmaps()`. După crearea imaginii mip-map a texturii, operaţiile de definire şi aplicaţie a texturii sunt asemănătoare cu cele prezentate în exemplele precedente. Setarea tipului de filtrare se face în funcţia `Display()` pentru fiecare din obiecte generate, prin funcţiile `glTexParameterf()`. În Fig. 6.8 se observă diferenţa dintre imaginea filtrată şi imaginea nefiltrată.



(a) (b)

Fig. 6.8 (a) Imagine cu textură nefiltrată.
(b) Imagine cu textură filtrată.

Creare imaginilor cu texturi cu filtrare mip-map necesită o putere de calcul deosebit de ridicată şi este necesară implementarea hardware în aceleratoarele grafice a algoritmilor de texturare pentru redarea în timp real.

7

Modele de reflexie şi iluminare

Redarea obiectelor tridimensionale prin suprafeţe colorate uniform crează imagini nerealiste şi dificil de interpretat. De exemplu, dacă se redă imaginea unei sfere aproximate prin poligoane colorând feţele vizibile cu aceeaşi culoare albă, se obţine un cerc alb. Acest aspect nerealist apare datorită faptului că percepţia celei de-a treia dimensiuni este mult influienţată de modul de redare a iluminării obiectelor. Imaginea aceleiaşi sferei în care se ţine seama de iluminare este mult mai sugestivă.

În lumea reală, atunci când lumina provenită de la diferite surse de lumină cade asupra obiectelor opace, o parte este absorbită de obiect, iar o parte este reflectată. Ochiul percepe lumina reflectată de obiect, pentru a interpreta forma, culoarea şi alte detalii ale obiectului. Pentru calculul iluminării în grafica pe calculator trebuie să fie definite sursele de lumină şi interacţiunea dintre lumină şi suprafeţe.

Un *model de iluminare* defineşte natura luminii emise de o sursă de lumină, adică distribuţia intensităţii luminii emise. Un *model de reflexie* descrie interacţiunea dintre lumină şi o suprafaţă, în funcţie de proprietăţile suprafeţei şi natura sursei de lumină. Modelele de iluminare şi de reflexie în grafica pe calculator permit redarea acceptabilă din punct de vedere al percepţiei umane a obiectele tridimensionale proiectate în spaţiul ecran bidimensional. Nivelul de acceptabilitate al redării depinde de natura aplicaţiei. Cu cât este cerut un nivel de realism mai ridicat, cu atât este necesar un model de reflexie mai complex şi cerinţe de prelucrare mai mari.

Implementarea unui model de reflexie în procedeul de calculare a intensităţii culorii fiecărui pixel este cunoscută sub numele de *tehnică de umbrire*.

Consideraţii teoretice asupra reflexiei luminii

Lumina incidentă la o suprafaţă a unui obiect este distribuită în patru categorii: lumina refectată, lumina adsorbită, lumina transmisă şi lumina împrăştiată şi emisă (Fig. 7.1).

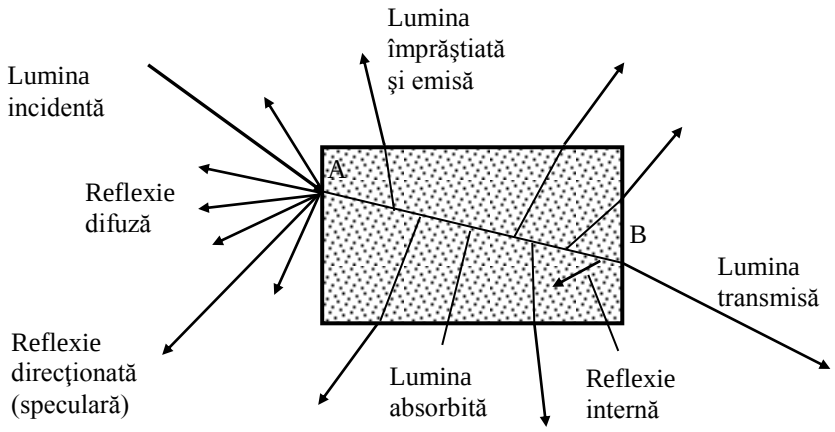


Fig. 7.1 Interacţiunea luminii cu un corp solid

În sinteza de imagine se încearcă modelarea cât mai eficientă şi completă a acestui mod de interacţiune a luminii cu obiectele. Intensitatea şi lungimea de undă a luminii reflectate de o suprafaţă a unui obiect depinde de mai mulţi factori: lungimea de undă a luminii incidente, unghiul de incidenţă, natura suprafeţei şi proprietăţile ei electrice: permitivitatea, permeabilitatea şi conducţanţa. Modelul exact al interacţiunii este extrem de complex şi el poate fi aproximat prin intermediul *funcţiei de reflexie bidirecţională* (*bidirectional reflectivity function* – BDRF), care permite o apreciere cantitativă suficient de sugestivă pentru redarea iluminării obiectelor în grafica pe calculator.

Funcţia de reflexie bidirecţională este relaţia dintre intensitatea luminii reflectate în direcţia ($\phi_v\,\theta_v$) şi energia luminii primite din direcţia ($\phi_i\,\theta_i$) (Fig. 7.2):

$$R_{bd}(\lambda,\phi_i,\theta_i,\phi_v,\theta_v)=\frac{I_V(\phi_i,\theta_i,\phi_v,\theta_v)}{E_i(\phi_i,\theta_i)} \qquad (7.1)$$

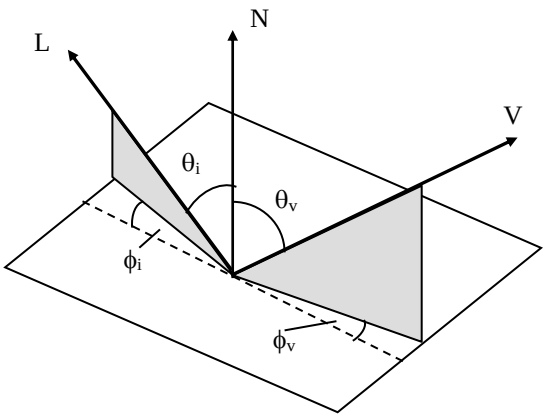


Fig. 7.2 Funcţia de reflexie bidirecţională

Funcţia de reflexie bidirecţională depinde de lungimea de undă λ. Relaţia între energia luminii incidente corespunzătoare unui unghi solid ω_i şi intensitate I_i este:

$$E_i(\phi_i, \theta_i) = I_i(\phi_i, \theta_i)\cos\theta_i d\omega_i \qquad (7.2)$$

Pentru grafica pe calculator se poate considera în mod simplificat că intensitatea reflectată este compusă din trei componente: componenta de reflexie direcţionată (speculară), componenta de reflexie difuză direcţionată şi componenta de reflexie difuză ideală. Primele două componente se datorează reflexiei de prim ordin, iar componenta de reflexie difuză ideală se datorează reflexiilor multiple şi a reflexiilor subsuprafeţelor (Fig. 7.3).

Valorile componentelor intensităţilor reflectate depind de rugozitatea suprafeţei şi de lungimea de undă a luminii. Dacă suprafaţa ar fi oglindă perfectă, atunci singura componentă de lumină reflectată ar fi componenta direcţionată (speculară). Pentru suprafeţe reale, cu un anumit grad de rugozitate, componenta speculară se defineşte ca intensitatea reflectată în direcţia: $\phi_v = \phi_i$, $\theta_v = \theta_i$, considerată faţă de orientarea medie a suprafeţei. Intensitatea acestei componente depinde de rugozitatea suprafeţei: cu cât suprafaţa este mai netedă, cu atât componenta speculară este mai mare.

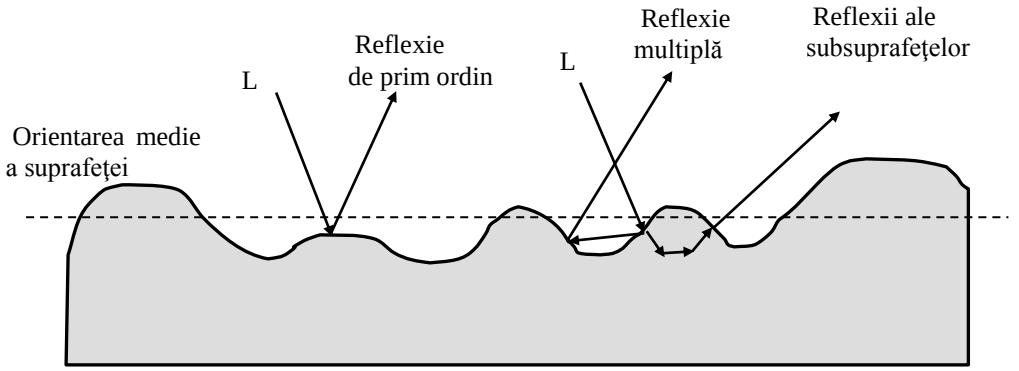


Fig. 7.3 Reflexii de prim ordin, reflexii multiple
şi reflexii ale subsuprafeţelor

Componenta de difuzie direcţionată depinde de asemenea de reflexiile de prim ordin. Direcţiile de reflexie sunt împrăştiate în semisfera centrată în punctul de

incidenţă, dar există o direcţie preferinţială, care este direcţia reflexiei speculare pentru suprafeţele cu rugozitate redusă.

Componenta de difuzie ideală reflectă lumina în toate direcţiile în interiorul unei semisfere centrate în punctul de incidenţă, datorită împrăştierii provocate de subsuprafeţele componente.

Prin descompunerea funcţiei de reflexie bidirecţională în trei componente se poate dezvolta un model analitic bazat pe aspecte fizice, optice şi geometrice, care să permită simularea iluminării şi a reflexiei în grafica pe calculator. Modelul de reflexie Phong este cel mai cunoscut model în grafica pe calculator, care adoptă o reprezentare empirică şi fenomenologică, prin formule de calcul simple de imitare a comportării teoretice a reflexiei luminii, descrise mai sus.

Modelul Phong de reflexie a luminii

Modelul de reflexie Phong imită eficient modul real de reflexie, până la un grad care produce o percepţie destul de bună obiectelor iluminate şi, de aceea, are o largă utilizare în grafica pe calculator. Modelul de reflexie Phong consideră lumina reflectată de o suprafaţă ca fiind o combinaţie liniară a trei componente: reflexia difuză, specular şi ambientală.

Componenta de reflexie difuză corespunde componentei de reflexie difuză ideală descrisă în secţiunea precedentă şi este evaluată prin relaţia:

$$I_d = I_i k_d (\mathbf{L} \bullet \mathbf{N}) = I_i k_d \cos\theta \qquad (7.3)$$

unde I_i este intensitatea luminii incidente, **L** este vectorul unitate îndreptat către sursa de lumină şi **N** este vectorul unitate normal la suprafaţă în punctul de incidenţă (Fig. 7.4(a)). Pentru o suprafaţă plană unghiul θ este constant şi, deci, componenta de reflexie difuză este constantă. k_d este un coeficient de reflexie (reflectanţă) empiric, care depinde de lungimea de undă a luminii şi de natura suprafeţei. Dat fiind că în reflexia difuză lumina este reflectată în toate direcţiile, această componentă nu depinde de poziţia de observare.

Dacă există mai multe surse de lumină, atunci:

$$I_d = k_d \sum_n I_{i,n} (\mathbf{L_n} \bullet \mathbf{N}) \qquad (7.4)$$

unde $\mathbf{L_n}$ este vectorul unitate în direcţia celei de-a n-a sursă de lumină.

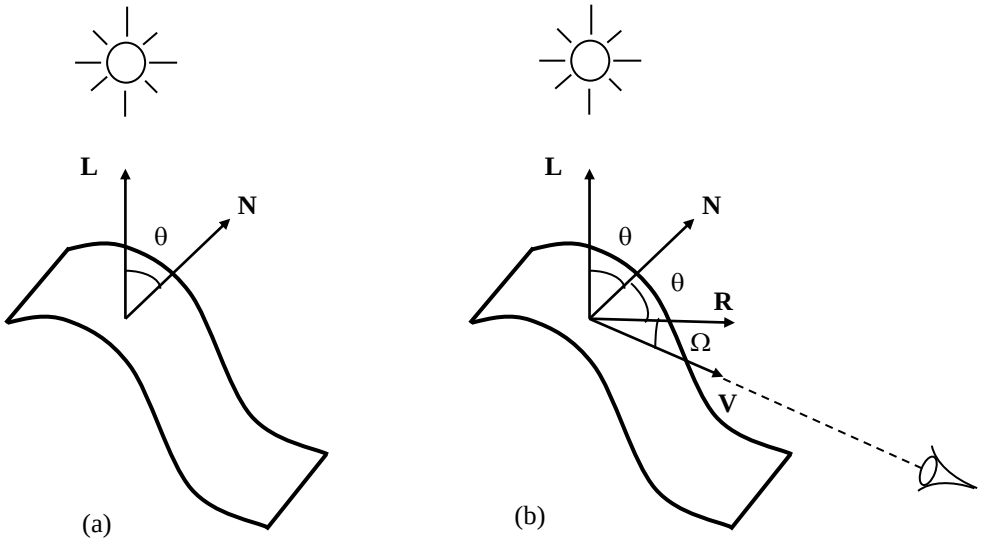


Fig. 7.4 (a) Reflexia difuză Phong.
(b) Reflexia speculară Phong.

Componenta speculară a reflexiei în modelul Phong depinde de unghiul Ω între direcţia de observare **V** şi direcţia de oglindire **R** (Fig. 7.4(b)). Dacă se consideră **R** şi **V** vectorii unitate ai acestor direcţii, atunci:

$$I_s = I_i k_s \cos^n \Omega = I_i k_s (\mathbf{R} \bullet \mathbf{V})^n \qquad (7.5)$$

unde n este un indice care simulează rugozitatea suprafeţei.

Pentru o suprafaţă oglindă perfectă, n tinde către infinit şi lumina este reflectată numai în direcţia de oglindire R, pentru care $\cos^n \Omega = 1$. Pentru valori finite ale indicelui n, se generează un lob de reflexie, a cărui grosime este o funcţie de rugozitatea suprafeţei (Fig. 7.5).

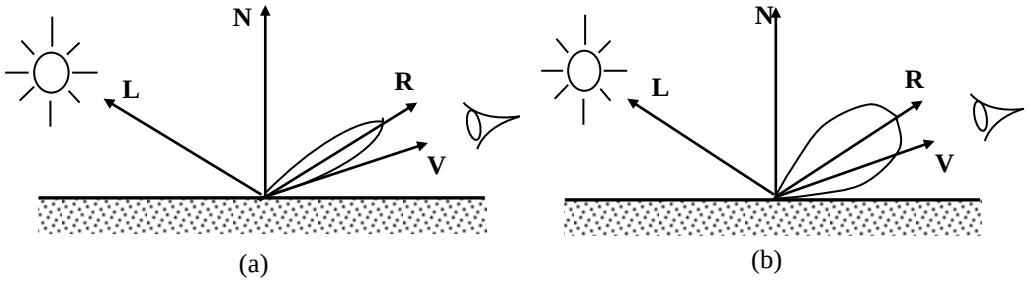


Fig. 7.5 (a) Indice n mare simulează reflexie speculară în lob îngust.
(b) Indice n mic simulează reflexie speculară în lob larg.

Efectul reflexiei speculare în modelul Phong este acela de a produce o iluminare mai accentuată (*highlight*), care este reflexia sursei de lumină pe o arie a suprafeţei, care depinde de valoarea lui n. Culoarea luminii reflectate specular poate fi diferită de culoarea luminii reflectate difuz. În modelele cele mai simple de reflexie speculară, se presupune că această componentă are culoarea sursei de lumină. De exemplu, o suprafaţă de culoare verde iluminată de o sursă de lumină albă produce o componentă de reflexie de difuzie de culoare verde, dar pata luminoasă de reflexie speculară are culoarea albă.

Componenta de lumină ambientală se adaugă în modelul Phong pentru a simula iluminarea globală a scenei. Această componentă permite iluminarea suprafeţelor care

nu sunt vizibile din sursa de lumină, dar sunt vizibile din punctul de observare. Fără componenta ambientală, suprafeţele care nu sunt îndreptate către o sursă de lumină ar fi redate complet negre, ceea ce ar crea un efect nerealist. Componenta ambientală aproximează iluminarea indirectă a suprafeţelor, de exemplu datorită reflexiilor multiple faţă de pereţii unei incinte. Aceste reflexii sunt foarte de complicat de simulat, astfel încât modelul Phong le ignoră şi adaugă o componentă de lumină ambientală constantă dată de relaţia:

$$I_g = I_a k_a \tag{7.6}$$

Prin însumarea componentelor de lumină difuză, speculară şi ambientală, se obţine iluminarea totală a unei suprafeţe:

$$I = I_a k_a + I_i \left(k_d (\mathbf{L} \bullet \mathbf{N}) + k_s (\mathbf{R} \bullet \mathbf{V})^n \right) \tag{7.7}$$

În modelul RGB de reprezentare a culorilor, relaţia 7.7 se descompune în trei relaţii, pentru fiecare componentă roşu, verde, albastru:

$$\begin{aligned} I_r &= I_a k_{ar} + I_i \left(k_{dr} (\mathbf{L} \bullet \mathbf{N}) + k_{sr} (\mathbf{R} \bullet \mathbf{V})^n \right) \\ I_g &= I_a k_{ag} + I_i \left(k_{dg} (\mathbf{L} \bullet \mathbf{N}) + k_{sg} (\mathbf{R} \bullet \mathbf{V})^n \right) \\ I_b &= I_a k_{ab} + I_i \left(k_{db} (\mathbf{L} \bullet \mathbf{N}) + k_{sb} (\mathbf{R} \bullet \mathbf{V})^n \right) \end{aligned} \tag{7.8}$$

În mod obişnuit se consideră componenta speculară de culoare albă (deci $k_{sr} = k_{sg} = k_{sb} = k_s$), iar culoarea suprafeţei este controlată prin coeficienţii de reflexie difuză (k_{dr}, k_{dg}, k_{db}) şi ambientală (k_{ar}, k_{ag}, k_{ab}). Rezultă relaţia de calcul a culorii unei suprafeţe în modelul Phong:

$$I(r,g,b) = I_a k_a (r,g,b) + I_i \left(k_d (r,g,b) (\mathbf{L} \bullet \mathbf{N}) + k_s (\mathbf{R} \bullet \mathbf{V})^n \right) \tag{7.9}$$

Modelul Phong de reflexie este un model foarte simplificat al comportării obiectelor iluminate. În acest model, se consideră sursele de lumină punctiforme, ignorându-se distribuţia surselor luminoase şi simulându-se un termen global de iluminare ambientală. Cu toate aceste simplificări, modelul Phong produce imagini cu un grad de realism care este suficient pentru multe aplicaţii.

Modele de umbrire

Aplicarea directă a relaţiei 7.8 pentru calculul culorii fiecărui pixel necesită un timp de execuţie extrem de ridicat, care nu este acceptabil în grafica interactivă. De aceea, în sinteza de imagine se folosesc anumite metode simplificate de calcul a culorii fiecărui pixel, numite tehnici (modele) de umbrire. Tehnicile de umbrire depind de modul de reprezentare a obiectelor. Pentru obiectele modelate prin reţea de poligoane, se folosesc mai multe modele de umbrire: umbrirea constantă (poligonală, *flat*), umbrirea Gouraud şi umbrirea Phong.

Modelul de umbrire constantă

În umbrirea constantă se admite calculul unei singure intensităţi a culorii pentru fiecare suprafaţă poligonală; acest mod de calcul este posibil dacă se admite ipoteza simplificatoare că atât sursele de lumină cât şi observatorul sunt la infinit. Umbrirea constantă produce discontinuităţi de culoare la frontiera dintre suprafeţe. Calculul umbririi constante înseamnă aplicarea formulelor Phong (7.8) pentru fiecare suprafaţă vizibilă, o singură dată pentru fiecare poziţie a observatorului.

Modelul de umbrire Gouraud

Pentru eliminarea discontinuităţilor de colorare care apar în umbrirea constantă, Gouraud (1971) a introdus o metodă de umbrire care-i poartă numele şi care calculează intensitatea de culoare a pixelilor suprafeţelor prin metode de interpolare, pornind de la intensităţile în vârfurile poligonului. Metoda incrementală de calcul a intensităţii de culoare a pixelilor este asemănătoare metodei de calcul a adâncimii pixelilor folosită în algoritmul Z-buffer. În general, se aplică combinat transformarea de rastru, eliminarea suprafeţelor ascunse şi umbrirea Gouraud, într-un algoritm de baleiere pe linii generalizat.

Umbrirea Gouraud este o tehnică de interpolare biliniară a intensităţii culorii, foarte simplă şi economică, care atenuează discontinuităţile de la frontiera poligoanelor prin care este reprezentat un obiect tridimensional, fără să elimine complet aspectul poligonal al obiectelor.

Intensitatea de culoare se calculează în vârfurile poligoanelor prin aplicarea relaţiilor 7.8 de calcul a reflexiei în modelul Phong. Normala într-un vârf al unui poligon se calculează ca medie a normalelor tuturor poligoanelor care sunt adiacente vârfului (Fig. 7.6):

$$N_v = \sum_{i=1}^{m} \frac{N_i}{m} \tag{7.10}$$

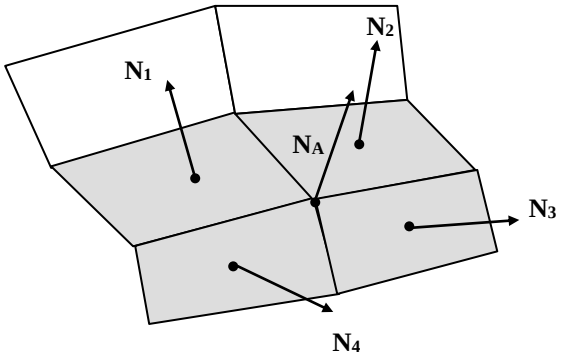


Fig. 7.6 Normala la vârful $\mathbf{N_A}$ este media normalelor $\mathbf{N_1}$, $\mathbf{N_2}$, $\mathbf{N_3}$, $\mathbf{N_4}$ a poligoanelor adiacente în vârful respectiv.

Normalele în vârfurile poligoanelor sunt definite în sistemul de referinţă model şi ele sunt transformate în sistemul de referinţă de observare prin aplicarea aceloraşi transformări care se aplică vârfurilor: transformarea de instanţiere şi transformarea de observare. În sistemul de referinţă de observare se calculează intensităţile în vârfurile poligoanelor şi aceste valori sunt folosite pentru interpolare biliniară în algoritmul de conversie de baleiere pe linii a poligoanelor.

Acest mod de calcul permite ca normalele în vârfuri să fie calculate o singură dată, la modelare, şi memorate ca parte a modelului obiectului în baza de date grafică. Atunci când se decupează feţele obiectelor, pot să apară vârfuri noi, care nu existau în modelul obiectului iniţial. Pentru aceste vârfuri se calculează normalele în noile vârfuri prin interpolare între normalele laturilor intersectate de planul de decupare.

Interpolarea biliniară a intensităţilor de culoare a pixelilor unui poligon se execută în cadrul algoritmului de baleiere pe linii pornind de la intensităţile de culoare ale vârfurilor poligonului, calculate prin relaţiile modelului de reflexie Phong.

Se reia exemplul de baleiere pe linii din secţiunile 5.3 şi 5.4.3. Poligonul ABCDE are intensităţile de culoare calculate în vârfuri: I_A, I_B, I_C, I_D, şi, respectiv, I_E. Intensităţile la capetele liniilor de baleiere se calculează din intensităţile vârfurilor (Fig. 7.7). Pentru laturile AB şi BC, ecuaţiile de interpolare sunt:

$$I = m_{i1}y+n_{i1}$$
$$I = m_{i2}y+n_{i2} \qquad (7.11)$$

unde: $m_{i1} = (I_A-I_B)/(y_A-y_B)$, $n_{i1} = y_B-m_{i1}\, I_B$

$$m_{i2} = (I_C-I_B)/(y_C-y_B),\ n_{i2} = y_C-m_{i2}\, I_C \qquad (7.12)$$

Din motive de eficienţă a calculelor, aceste ecuaţii se implementează incremental. Intensităţile se calculează în punctele de intersecţie cu liniile de baleiere y_1, y_2, …y_i, y_{i+1},…Fiind calculate intensităţile $I_{i,1}$ şi $I_{i,2}$ corespunzătoare liniei de baleiere $y = y_i$, intensităţile $I_{i+1,1}$ şi $I_{i+1,2}$, corespunzătoare liniei de baleiere următoare, $y_{i+1} = y_i+1$ se obţin prin incrementare:

$$I_{i+1,1} = I_{i,1} + m_{i1}$$
$$I_{i+1,2} = I_{i,2} + m_{i2} \qquad (7.13)$$

Intensitatea $I_{i,1}$ se atribuie pixelului a cărui adresă se obţine din coordonatele punctului $(x_{i,1}, y_i)$, intensitatea $I_{i,2}$ se atribuie pixelului a cărui adresă se obţine din coordonatele punctului $(x_{i,2}, y_i)$, etc.

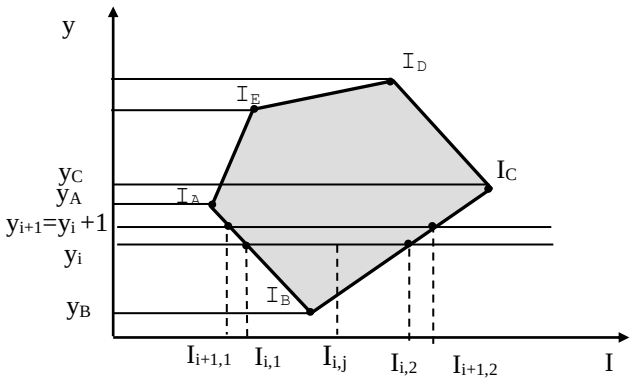


Fig. 7.7 Interpolarea intensităţii pe laturile poligonului şi pe o linie de baleiere în umbrirea Gouraud.

Intensităţile de-a lungul liniei de baleiere se calculează din intensităţile la capetele acesteia. Pentru linia de baleiere $y = y_i$, ecuaţia de interpolare a intensităţii este:

$$I = m_{ix}x+n_{ix}$$

unde: $m_{ix} = (I_{i,2}-I_{i,1}) / (x_{i,2}-x_{i,1})$, $n_{ix} = x_{i,1}-m_{ix}\, I_{i,1}$ (7.14)

Implementarea eficientă a acestei interpolări se face de asemenea incremental. Fiind calculată intensitatea $I_{i,j}$ corespunzătoare punctului de coordonate $(x_{i,j}, y_i)$, intensitatea în punctul următor eşantionat pe linia de baleiere $(x_{i,j+1}, y_i$, unde $x_{i,j+1} = x_{i,j}+1)$ este:

$$I_{i,j+1} = I_{i,j} + m_{ix} \quad (7.15)$$

Pentru fiecare linie de baleiere se calculează intensitatea în puncte succesive prin incrementarea valorii precedente cu coeficientul dat de relaţia 7.14.

În Fig. 7.8 este exemplificată diferenţa dintre umbrirea poligonală şi umbrirea Gouraud. Două sfere reprezentate prin reţea de suprafeţe plane (poligoane) sunt iluminate de la aceeaşi sursă punctiformă de lumină şi au aceleaşi caracteristici de reflectanţă ai materialului. Diferenţa dintre cele două imagini ale sferei provine din faptul că pentru prima sferă se foloseşte umbrirea poligonală iar pentru cea de-a doua sferă umbrirea Gouraud.

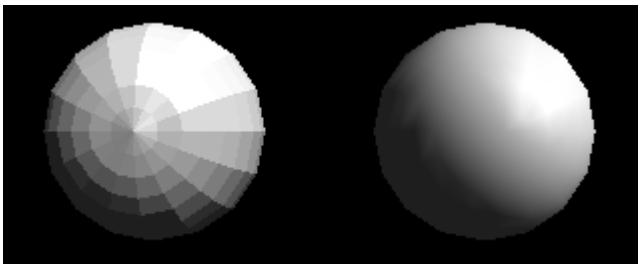
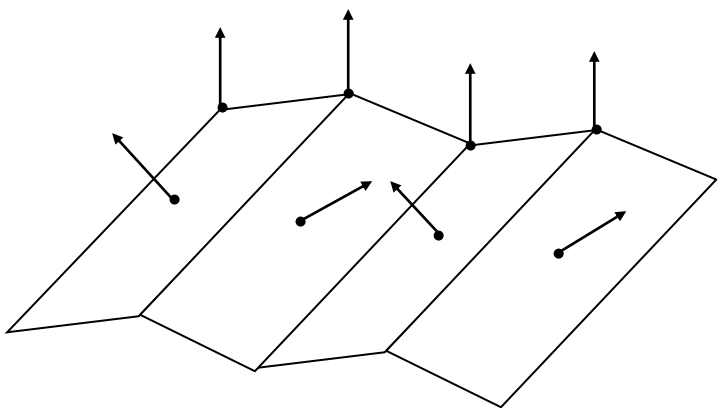


Fig. 7.8 Diferenţa dintre umbrirea poligonală şi umbrirea Gouraud.

Deşi acceptabilă în multe aplicaţii, atât ca eficienţă cât şi ca realism de reprezentare, umbrirea Gouraud poate să provoace unele anomalii care provin în primul rând din calculul normalei într-un vârf prin medierea normalelor feţelor adiacente. De exemplu, în Fig. 7.9 se observă cum este posibil să se obţină normale în vârfuri identice (şi deci intensităţi egale), din normale la suprafeţe diferite.



O altă deficienţă a modelului Gouraud este că nu se poate obţine o pată de lumină (highlight) datorată reflexiei speculare în interiorul unui poligon, dacă vârfurile poligonului nu sunt cuprinse în această pată de lumină.

Fig. 7.9 O suprafaţă plată regulat produce normale în vârfuri egale, deci colorare constantă a suprafeţelor în umbrirea Gouraud.

O parte din deficienţele tehnicii de umbrire Gouraud sunt eliminate în modelul de umbrire Phong.

Modelul de umbrire Phong

Modelul de umbrire Phong este de asemenea un model de calcul a intensităţii culorii prin interpolare biliniară, ca şi modelul Gouraud, dar în acest model se interpolează normalele şi se calculează exact intensitatea culorii (Fig. 7.10).

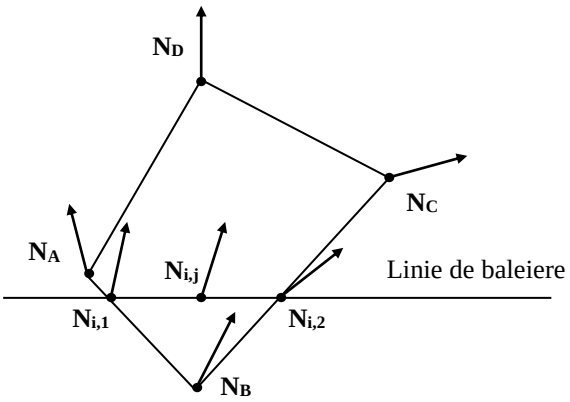


Fig. 7.10 Interpolarea normalelor în modelul de umbrire Phong.

Etapele de calcul a intensităţii culorii în modelul de umbrire Phong sunt:

- Calculul normalelor în vârfurile poligonului.
- Pentru fiecare linie de baleiere, se calculează prin interpolare vectorii normali ai intersecţiei liniei de baleiere cu laturile poligonului, folosind normalele în vârfuri ($\mathbf{N_{i,1}}$ şi $\mathbf{N_{i,2}}$ în figura de mai sus).
- Normalele la capetele unei linii de baleiere sunt folosite pentru calculul prin interpolare a normalei fiecărui punct (căruia în corespunde un pixel) de pe linia de baleiere ($\mathbf{N_{i,j}}$).

- Normalele calculate prin interpolare sunt folosite pentru calculul intensităţii culorii în fiecare punct.

Umbrirea Phong este mai costisitoare din punct de vedere al operaţiilor efectuate deoarece, pentru fiecare pixel, se aplică relaţia de calcul 7.8, care conţine produse scalare de vectori şi înmulţiri. Dar, din punct de vedere al calităţii imaginii generate, umbrirea Phong este mai bună, dat fiindcă se calculează normale foarte apropiate de normala reală în fiecare punct al suprafeţei. În tehnica de umbrire Phong se poate simula mult mai corect reflexia speculară (*highlight*).

În secţiunea următoare sunt prezentate modalităţile de programare a iluminării în aplicaţii grafice folosind biblioteca OpenGL.

Funcţiile OpenGL de calcul a iluminării

Biblioteca OpenGL implementează modelul Phong de reflexie şi modelele de umbrire poligonală şi umbrire Gouraud.

În mod implicit, sistemul de iluminare este inactivat şi pentru desenarea primitivelor geometrice se foloseşte culoarea curentă, specificată prin funcţia `glColor#()`. Pentru validarea iluminării obiectelor, se activează sistemul de iluminare OpenGL prin apelul funcţiei `glEnable(GL_LIGHTING)`.

Pentru calculul iluminării obiectelor trebuie să fie definite:

- sursele de lumină;
- materialul suprafeţelor;
- modelul de umbrire a suprafeţelor.

Definirea surselor de lumină

În OpenGL se pot defini mai multe surse de lumină punctiforme. Numărul de lumini admise variază în funcţie de implementare, dar cel puţin opt lumini sunt disponibile în orice bibliotecă OpenGL. Fiecare sursă de lumină poate fi validată prin apelul funcţiei `glEnable(GL_LIGHTi)`, unde i este indexul sursei de lumină.

O sursă de lumină se caracterizează prin intensitate şi poziţie în scenă. Intensitatea unei surse de lumină se specifică pentru fiecare componentă de iluminare (ambientală, de difuzie şi speculară) printr-un vector în spaţiul culorilor în modelul RGBA. Poziţia unei surse se specifică printr-un vector în coordonate omogene corespunzătoare sistemului de referinţă universal.

Funcţia de definire a unui parametru al unei surse de lumină este funcţia `glLight#()`, care are mai multe variante în funcţie de tipul argumentelor. De exemplu:

```
void glLightfv(GLenum light, GLenum pname,
                                    const GLfloat *params);
void glLightiv(GLenum light, GLenum pname,
                          const GLint *params);
```

Argumentul `light` reprezintă indexul sursei de lumină şi poate lua un nume simbolic de forma `GL_LIGHT0, GL_LIGHT1, ….GL_LIGHTi,` unde $0 \leq i <$ `GL_MAX_LIGHTS`. Numărul maxim de lumini depinde de implementarea bibliotecii.

Argumentul `pname` specifică un parametru al sursei de lumină. Sunt acceptate mai multe valori, dintre care unele se referă la intensitate de culoare iar altele la poziţia sursei.

Valorile `GL_AMBIENT`, `GL_DIFFUSE`, `GL_SPECULAR` ale argumentului `pname` permit definirea componentelor intensităţii culorii sursei de lumină. În această situaţie, argumentul `params` reprezintă un pointer la un vector de patru valori (de tip întreg sau virgulă mobilă) care specifică intensităţile RGBA ale componentei de iluminare ambientală, difuză şi, respectiv, speculară. Valorile implicite ale intensităţilor sunt (`0.0,0.0,0.0,1.0`) pentru componenta ambientală şi pentru componentele difuză şi speculară a oricărei lumini cu excepţia luminii cu index 0, care au intensitatea difuză şi speculară implicită (`1.0,1.0,1.0,1.0`).

În acest model de definire a sursei de lumină, componenta ambientală (`GL_AMBIENT`) se referă la intensitatea RGBA pe care o sursă de lumină o adaugă iluminării globale a scenei. Componenta de iluminare difuză (`GL_DIFFUSE`) este cel mai apropiat mod de reprezentare a ceea ce se consideră culoarea sursei de lumină. Componenta de iluminare speculară (`GL_SPECULAR`) afectează culoarea zonei strălucitoare (*hightlight*) a obiectelor luminate de sursa de lumină respectivă. Valoarea transparenţei (*alpha*) surselor de lumină este ignorată dacă nu se validează calculul transparenţei (*blending*).

Valoare `GL_POSITION` a argumentului `pname` permite definirea poziţiei sursei de lumină. În această situaţie, argumentul `params` este un pointer la un vector de patru numere (întregi sau virgulă flotantă) care reprezintă poziţia în coordonate omogene în sistemul de referinţă universal a sursei de lumină. Acestă poziţie este transformată prin aplicarea valorii matricii de modelare-vizualizare din momentul apelului funcţiei `glLight`, astfel încât sursa de lumină va avea coordonatele transformate în sistemul de referinţă de observare, unde se calculează intensitatea în vârfurile primitivelor geometrice.

Dacă componenta `w` a poziţiei este 0, atunci lumina este tratată ca o lumină direcţională plasată la infinit, în direcţia definită de componentele `x,y,z` şi se utilizează această direcţie pentru calculul componentelor de reflexie difuză şi speculară.

Dacă $w \neq 0$, atunci sursa este o sursă poziţională şi se foloseşte localizarea acesteia pentru calculul direcţiei de iluminare a suprafeţelor. Poziţia implicită a unei surse este (`0,0,1,0`), deci sursa este plasată la infinit pe axa z şi ea luminează obiectele în direcţia – z.

Secvenţa de instrucţiuni pentru definirea intensităţii şi poziţiei sursei de lumină de indice 0 arată astfel:

```
GLfloat light_ambient[] = {1.0, 0.0, 0.0, 0.0};
GLfloat light_diffuse[] = {1.0, 0.0, 0.0, 0.0};
GLfloat light_specular[] = {1.0, 1.0, 1.0, 0.0};
GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

În iluminarea reală, intensitatea luminii primite de un obiect scade odată cu creşterea distanţei acestuia faţă de sursa de lumină. Dat fiind că sursele de lumină direcţionale (care au `w =0`) se consideră plasate la infinit, nu are sens calculul atenuării cu distanţa, astfel încât atenuarea este invalidată pentru astfel de surse. Pentru sursele poziţionale, se foloseşte un factor de atenuare cu valoarea:

$$f = \frac{1}{k_c + k_l d + k_q d^2} \tag{7.16}$$

unde: `d` este distanţa dintre sursa de lumină şi vârful în care se calculează iluminarea
k_c = `GL_CONSTANT_ATTENUATION`
k_l = `GL_LINEAR_ATTENUATION`
k_q = `GL_QUADRATIC_ATTENUATION`

Implicit, aceşti parametri au valorile: $k_c=1$, $k_l=0$, $k_q=0$, dar ei pot fi setaţi prin apelul uneia din funcţiile `glLightf()` sau `glLighti()` astfel:

```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 1.0);
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.5);
```

Dacă se folosesc mai multe surse de lumină, pentru fiecare dintre ele se definesc componentele intensităţii, poziţia şi caracteristicile de atenuare.

Definirea proprietăţilor materialelor

Pentru calculul intensităţii culorii unei suprafeţe, trebuie să fie definite proprietăţile de reflexie a luminii pentru componentele de reflexie ambientală, difuză şi specular. Aceste proprietăţi sunt considerate proprietăţi de material al suprafeţei şi se specifică prin apelul uneia sau mai multora din cele patru variante ale funcţiei `glMaterial#()`:

```
void glMateriali(GLenum face,GLenum pname,GLint param);
void glMaterialf(GLenum face,GLenum pname,GLfloat param);
void glMaterialiv(GLenum face,GLenum pname,GLint *param);
void glMaterialfv(GLenum face,GLenum pname,
        GLfloat *param);
```

În aceste funcţii, argumentul `face` este numele feţei şi depinde de orientarea acesteia; poate lua ca valori constantele simbolice `GL_FRONT`, `GL_BACK`, `GL_FRONT_AND_BACK`. Argumentul `pname` specifică proprietatea materialului care se defineşte prin apelul funcţiei `glMaterial#()`. Acest argument poate lua una valorile date în tabelul de mai jos.

Tabelul 7.1 Parametrii de definire a materialelor

Nume parametru	Valoare implicită	Semnificaţie
GL_AMBIENT	(0.2,0.2,0.2,1.0)	Reflectanţa (coeficient de reflexie) ambientală
GL_DIFFUSE	(0.8,0.8,0.8,1.0)	Reflectanţa de difuzie

GL_AMBIENT_AND _DIFFUSE		Reflectanţa ambientală şi de difuzie
GL_SPECULAR	(0.0,0.0,0.0,1.0)	Reflectanţa specular
GL_EMISSION	(0.0,0.0,0.0,1.0)	Intensitatea luminii emise
GL_SHININESS	0.0	Exponentul de reflexie specular
GL_COLOR_INDEX	(0,1,1)	Indicii culorilor ambientală, difuză şi specular ale materialului

Funcţiile `glMateriali()` şi `glMaterialf()` se folosesc numai pentru setarea strălucirii (`GL_SHININESS`) la valoarea dată prin argumentul `param`.

Funcţiile `glMaterialiv()` şi `glMaterialfv()` se folosesc pentru specificarea celorlalte proprietăţi. Dacă argumentul `pname` este `GL_COLOR_INDEX`, atunci `param` este un pointer la un vector de trei valori de tip întreg, care conţine indicii culorilor ambientală, de difuzie şi specular ale materialului. În celelalte situaţii, `param` este un pointer la un vector de patru valori de tip întreg sau flotant, care sunt componentele roşu, verde, albastru şi alpha (transparenţă) ale componentei de reflexie specificată prin parametrul `pname`(ambientală, difuză, specular, emisivitate). Se poate observa faptul că modelul OpenGL de reflexie permite adăugarea unei componente emisive a luminii reflectată de o suprafaţă.

Reflexia difuză joacă cel mai important rol în culoarea pe care o prezintă o suprafaţă. Ea reprezintă culoarea pe care o are suprafaţa luminată direct şi depinde de componenta de difuzie a luminii incidente, provenită de la una sau mai multe surse de lumină, de coeficientul de reflexie de difuzie (reflectanţa de difuzie) a materialului şi de unghiul dintre direcţia luminii şi normala la suprafaţă. Poziţia de observare nu influienţează componenta de difuzie a luminii reflectate.

Reflexia ambientală afectează culoarea de ansamblu pe care o prezintă o suprafaţă şi ea devine sesizabilă atunci când suprafaţa nu este luminată direct. Ca şi reflexia difuză, această componentă nu depinde de poziţia de observare. Cele două componente se specifică de cele mai multe ori cu aceeaşi culoare (aşa cum sunt suprafeţele reale) folosind parametrul `GL_AMBIENT_AND_DIFFUSE` în apelul funcţiei `glMaterial#()`.

Reflexia specular produce iluminarea mai puternică (*highlight*) a unei zone a obiectului, în funcţie de poziţia de observare. OpenGL permite specificarea culorii produse de reflexia specular (prin parametrul `GL_SPECULAR`) şi dimensiunea şi strălucirea zonei prin parametrul `GL_SHININESS`.

Componenta de emisie a unei suprafeţe se specifică prin parametrul `GL_EMISSION`. Acest efect este folosit pentru simularea lămpilor sau a altor surse de lumină din scenă.

Proprietăţilor materialelor. Funcţia `glMaterial#()` defineşte proprietăţile materialului curent, care se aplică tuturor vârfurilor introduse după aceasta prin funcţiile `glVertex#()` sau prin diferite funcţii de modelare din biblioteca `GLUT` (de exemplu, `glutSolidSphere()`). Proprietăţile materialului curent se menţin până la următorul apel al funcţiei `glMaterial#()`. Calculele de iluminare se pot executa diferit pentru feţele orientate direct (`GL_FRONT`) şi cele orientate invers (`GL_BACK`).

Un exemplu de definire a unui material este dat prin următoarele instrucţiuni:

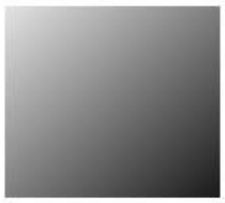
```
GLfloat mat_ambient[] = {1.0, 0.0, 0.0, 1.0};
GLfloat mat_diffuse[] = {1.0, 0.0, 0.0, 1.0};
GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};
GLfloat mat_shininess = 50.0;
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);
```

Modelul de umbrire se defineşte prin apelul funcţiei `glShadeModel(GLenum mode)`, unde argumentul mode poate lua una din valorile `GL_FLAT`, pentru modelul de umbrire poligonală, sau `GL_SMOOTH`, pentru modelul de umbrire Gouraud. Valoarea implicită este `GL_SMOOTH`.

Dacă nu este validat sistemul de iluminare (prin apelul funcţiei `glEnable(GL_LIGHTING);`) atunci culoarea care se atribuie vârfurilor primitivelor geometrice este culoarea curentă, setată prin apelul unei funcţii `glColor#()`. Dacă s-a definit modelul de umbrire poligonală (`GL_FLAT`), primitivele geometrice se generează de culoare constantă (culoarea curentă întâlnită la primul vârf introdus prin funcţia `glVertex#()`). Dacă s-a definit modelul de umbrire Gouraud (`GL_SMOOTH`), atunci culorile definite în vârfuri se folosesc pentru calculul intensităţii culorii pixelilor primitivei prin interpolarea biliniară.

ν Exemplul 7.1

În Fig 7.11 (a) s-a definit umbrirea Gouraud (`GL_SMOOTH`), iar în Fig 7.11 (b) s-a definit umbrirea poligonală (`GL_FLAT`), toate celelate funcţii fiind aceleaşi.



```
glShadeModel(GL_SMOOTH);
glBegin(GL_POLYGON);
      glColor3f(0.5,0.5,0.5);
            glVertex3d(-1,-1,-8);
            glColor3f(0.1,0.1,0.1);
            glVertex3d(1,-1,-8);
            glColor3f(0.5,0.5,0.5);
            glVertex3d(1,1,-8);
            glColor3f(0.9,0.9,0.9);
            glVertex3d(-1,1,-8);
glEnd();
```

(a)

```
glShadeModel(GL_FLAT);
glBegin(GL_POLYGON);
      glColor3f(0.5,0.5,0.5);
            glVertex3d(-1,-1,-8);
            glColor3f(0.1,0.1,0.1);
            glVertex3d(1,-1,-8);
            glColor3f(0.5,0.5,0.5);
            glVertex3d(1,1,-8);
            glColor3f(0.9,0.9,0.9);
            glVertex3d(-1,1,-8);
glEnd();
```

(b)

Fig. 7.11 (a) Interpolarea biliniară a culorilor din vârfurile primitivei.
(b) Colorare constantă a primitivei.

ν **Exemplul 7.2**

Pentru generarea imaginii din Fig. 7.8 din secţiunea precedentă, s-a validat sistemul de iluminare, s-a definit sursa de lumină `GL_LIGHT0` şi s-au definit parametrii materialului feţelor sferei. Programul scris folosind sistemul de dezvoltare `GLUT` este următorul:

```
#include <GL/glut.h>

void Init(){
        GLfloat light_ambient[] = {1.0, 1.0, 1.0, 1.0};
        GLfloat light_diffuse[] = {1.0, 1.0, 1.0, 1.0};
        GLfloat light_specular[] = {1.0, 1.0, 1.0, 1.0};
        GLfloat mat_ambient[] = {0.1, 0.1, 0.1, 1.0};
        GLfloat mat_diffuse[] = {0.9, 0.9, 0.9, 1.0};
        GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};
  GLfloat mat_shininess =  50.0 ;

        glClearColor(0.0,0.0,0.0,0.0);
  glEnable(GL_DEPTH_TEST);
        glClearDepth(1.0f);                     // implicit
        glDepthFunc(GL_LESS);                   // implicit
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
        glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
        glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
        glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
        glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
  glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
  glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
  glMaterialf(GL_FRONT, GL_SHININESS, mat_shininess);
        glShadeModel(GL_SMOOTH);                // implicit
}
void Display(void){
        GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};
        glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glLightfv(GL_LIGHT0, GL_POSITION, light_position);
        glPushMatrix();
                glTranslatef(-1.5,0,-8);        //transf. modelare
                glShadeModel(GL_FLAT);
```

```
                glutSolidSphere(1,16,16);
        glPopMatrix();
        glPushMatrix();
                glTranslatef(1.5,0,-8);                 // transf. modelare
                glShadeModel (GL_SMOOTH);
                glutSolidSphere(1,16,16);
        glPopMatrix();
        glutSwapBuffers();
}
void Reshape(int w, int h){
  h = (h == 0) ? 1 : h;
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(60.0,(GLfloat)w/(GLfloat)h, 1, 40.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}
int main(int argc, char** argv){
  glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH );
        glutInitWindowSize(500,500);
        glutInitWindowPosition(100,100);
        glutCreateWindow("Lights");
        Init();
        glutDisplayFunc(Display);
        glutReshapeFunc(Reshape);
        glutMainLoop();
        return 0;
}
```

Funcţiile callback `Display()` şi `Reshape()` au aceleeaşi semificaţie care a fost descrisă în programele precedente. Sursa de lumină direcţională (plasată la distanţă infinită) are culoarea albă, iar materialul are componenta specular albă, iar componentele difuză şi ambientală au nuanţe de gri. Pentru redarea primei sfere s-a setat umbrire poligonală (`GL_FLAT`), iar pentru cea de-a doua sferă umbrire Gouraud (`GL_SMOOTH`). Imaginea din Fig. 7.11 evidenţiază diferenţa dintre umbrirea poligonală şi umbrirea Gouraud, precum şi iluminarea accentuată (pata luminoasă) datorată reflexiei speculare.

Dacă în programul de mai sus se înlocuiesc apelurile funcţiilor `glutSolidSphere(1,16,16)` cu apelul funcţiilor `glutSolidTorus(0.5, 0.9,8,16)` se obţine imaginea din Fig. 7.12, în care este şi mai pregnantă diferenţa dintre umbrirea poligonală şi umbrirea Gouraud.

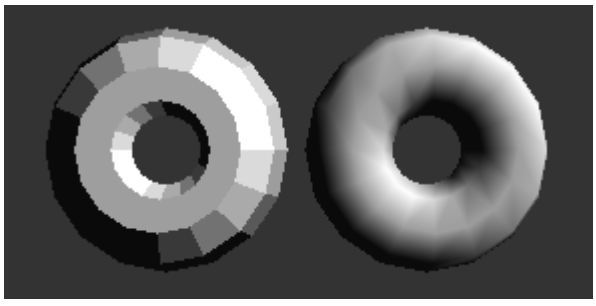


Fig. 7.12 Tor cu umbrire poligonală şi umbrire Gouraud.

ν

Normalele în vârfurile primitivelor geometrice

Pentru calculul reflexiei difuze şi speculare este necesar să fie definite normalele în vârfurile primitivelor geometrice. Normala care se foloseşte pentru calculul reflexiei într-un vârf este normala curentă, care se setează prin apelul unei variante a funcţiei `glNormal#()` şi se memorează într-o variabilă de stare a bibliotecii, identificată prin constanta simbolică `GL_CURRENT_NORMAL`. O parte din prototipurile funcţiilor `glNormal#()` definite în fişierul header `gl.h` sunt:

```
void glNormal3d(GLdouble nx, GLdouble ny, GLdouble nz);
void glNormal3f(GLfloat nx, GLfloat ny, GLfloat nz);
void glNormal3i(GLint nx, GLint ny, GLint nz);
void glNormal3bv(const GLbyte *v);
void glNormal3dv(const GLdouble *v);
void glNormal3fv(const GLfloat *v);
```

Normala se poate specifica prin componentele `nx, ny, nz,` sau printr-un pointer la un vector care conţine cele trei componente, de tipul cerut prin numele funcţiei `glNormal#()`.

Normalele se poate defini în orice loc în program, în exteriorul sau în interiorul unui bloc `glBegin()-glEnd()`. Dacă normala se defineşte o singură dată pentru o primitivă geometrică, această valoare este folosită pentru calculul intensităţilor tuturor vârfurilor primitivei geometrice. În mod normal însă, pentru calculul corect al intensităţilor culorii în vârfuri, este necesar ca normala să fie definită pentru fiecare vârf al primitivei geometrice, deci funcţiile `glNormal#()` se apelează alternant cu funcţiile `glVertex#()` în interiorul blocului `glBegin()-glEnd()`.

În exemplul precedent nu a fost necesară definirea normalelor, deoarece funcţia `glutSolidSphere()` le defineşte în mod automat, dar, pentru calculul iluminării unui obiect definit printr-o mulţime de primitive date prin vârfurile lor, este nesesar să fie definite toate normalele în vârfuri.

Normalele se specifică în aceelaşi sistem de referinţă ca şi vârfurile primitivelor geometrice, deci în sistemul de referinţă de modelare. Acest mod de calcul permite ca normalele în vârfuri să fie calculate o singură dată, la modelare, şi memorate ca parte a modelului obiectului în baza de date grafică. Asupra lor se aplică în mod automat transformarea de modelare-vizualizare folosind valoarea matricei din vârful stivei de modelare-vizualizare, existentă în momentul apelului funcţiei `glNormal#()`. În felul acesta, normalele sunt transformate în sistemul de referinţă de observare, unde se calculează intensităţile componentelor de reflexie în vârfurile primitivelor geometrice.

ν **Exemplul 7.3**

În acest exemplu se evidenţiază necesitatea definirii normalelor pentru calculul iluminării. În program este validat sistemul de iluminare şi este definită o sursă de lumină direcţională şi materialul, cu componente de reflectanţă difuză şi ambientală. Cubul din Fig. 7.13(a) a fost generat cu definirea normalelor în fiecare vârf al fiecărei suprafeţe. În Fig. 7.13 (b), singura diferenţă este absenţa normalelor în vârfuri; se observă absenţa umbririi obiectului.

(a) (b)

Fig. 7.13 (a) Calculul iluminării folosind normalele în vârfuri.
(b) Imaginea obţinută dacă nu sunt definite normalele în vârfuri.

Programul care produce aceste imagini este foarte asemănător cu programul din Exemplul 7.2 pentru toate funcţiile, mai puţin funcţia callback `Display()`, care arată astfel:

```
void Display(){
        glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glEnable (GL_NORMALIZE);
        glPushMatrix();
                glTranslated(-2,-2,-8);  // transf. de observare
                glLightfv(GL_LIGHT0, GL_POSITION, light_position);

                glBegin(GL_POLYGON);
                        glNormal3d(-1,1, 1);
                        glVertex3d(-1,1, 1);
                        glNormal3d( 1,1, 1);
                        glVertex3d( 1,1, 1);
                        glNormal3d( 1,1,-1);
                        glVertex3d( 1,1,-1);
                        glNormal3d(-1,1,-1);
                        glVertex3d(-1,1,-1);
                glEnd();
                glBegin(GL_POLYGON);
                        glNormal3d(-1,-1,1);
                        glVertex3d(-1,-1,1);
                        glNormal3d( 1,-1,1);
                        glVertex3d( 1,-1,1);
                        glNormal3d( 1, 1,1);
```

```
                glVertex3d( 1, 1,1);
                glNormal3d(-1, 1,1);
                glVertex3d(-1, 1,1);
        glEnd();
        glBegin(GL_POLYGON);
                glNormal3d(1,-1, 1);
                glVertex3d(1,-1, 1);
                glNormal3d(1,-1,-1);
                glVertex3d(1,-1,-1);
                glNormal3d(1, 1,-1);
                glVertex3d(1, 1,-1);
                glNormal3d(1, 1, 1);
                glVertex3d(1, 1, 1);
        glEnd();
        ……
    glPopMatrix();
    glutSwapBuffers();
}
```

În funcţia `Display()` sunt introduse numai trei din cele şase feţe ale cubului; celelate trei se definesc în mod similar. Funcţia `glEnable(GL_NORMALIZE)` are ca efect scalarea la valoarea unitate a vectorilor normalelor introduse prin funcţiile `glNormal#()`.

Dacă se comentează toate funcţiile `glNormal3d()` din programul de mai sus, se obţine imaginea din Fig. 7.13(b), în care nu se obţine umbrirea feţelor cubului.

Controlul poziţiei şi al direcţiei surselor de lumină

OpenGL tratează poziţia şi direcţia surselor de lumină la fel cum este tratată poziţia primitivelor geometrice, adică sursele de lumină sunt transformate cu aceleaşi matrici de transformare ca şi primitivele geometrice. Atunci când este apelată funcţia `glLight#(GL_LIGHTi, GL_POSITION vect)` pentru specificarea poziţiei sau direcţiei unei surse de lumină, poziţia sau direcţia este transformată de matricea de modelare-vizualizare curentă şi este memorată în coordonate în sistemul de referinţă de observare. Acest lucru înseamnă că se poate modifica poziţia sau direcţia unei surse de lumină prin modificarea corespunzătoare a matricii de modelare-vizualizare. Matricea de proiecţie nu modifică poziţia surselor de lunimă. Sursele de lumină pot fi tratate în trei moduri:

- surse cu poziţie fixă;
- surse care se deplasează odată cu punctul de observare;
- surse care se deplasează în scenă.

Surse de lumină cu poziţie fixă. Funcţia `glLight#()` cu parametrul `GL_POSITION` defineşte coordonatele omogene ale poziţiei sursei. Dacă această funcţie este apelată în momentul în care matricea curentă din stiva de modelare-vizualizare conţine matricea de observare, atunci poziţia sursei este transformată în coordonate de observare. Poziţia în spaţiu a sursei rămîne fixă, iar poziţia calculată şi utilizată în calcule este poziţia în coordonate de observare. În Exemplul 7.3 sursa de lumină este menţinută în poziţie fixă în scenă prin setarea poziţiei ei în momentul în

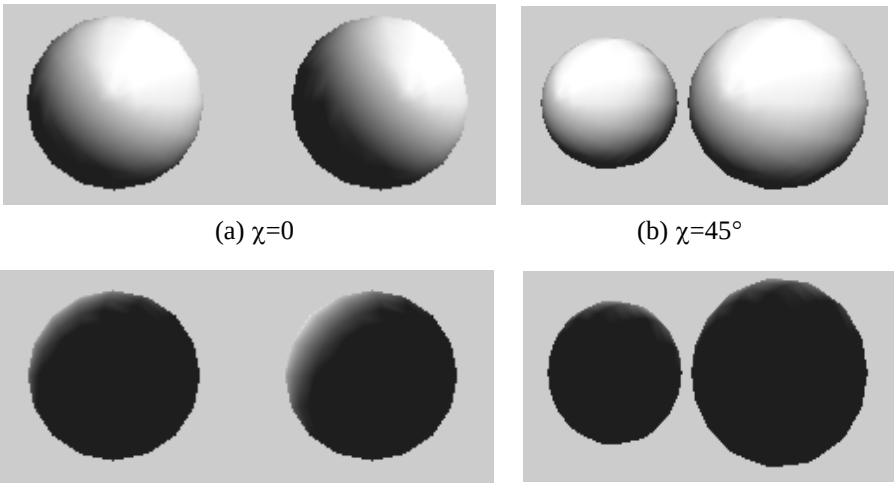
care matricea curentă de modelare-vizualizare conţine matricea de transformare de observare:

```
glPushMatrix();
        glTranslated(-2,-2,-8);  // transf. de observare
        glLightfv(GL_LIGHT0, GL_POSITION, light_position);
        // Desenare obiect
glPopMatrix();
```

ν **Exemplul 7.4**

Se reia programul de desenare a două sfere luminate din Exemplul 7.2. Culoarea de ştergere este modificată (culoare gri deschis (0.8, 0.8, 0.8)) pentru o mai bună evidenţiere a umbririi obiectelor. Transformările de instanţiere se aplică fiecărui obiect în parte. Transformarea de observare se aplică tuturor obiectelor şi direcţiei sursei de lumină. În continuare este prezentată numai funcţia `Display()`, pentru evidenţierea poziţiei fixe a sursei de lumină, în condiţiile modificării punctului de observare (x_V=0, y_V=0, z_V=8 şi unghiul χ varibil).

```
void Display(){
        GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};
        glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glPushMatrix();
                glTranslatef(0,0,-8);
                glRotated(gamma,0,1,0);                 // transf. observare
                glLightfv(GL_LIGHT0, GL_POSITION, light_position);
                glPushMatrix();
                        glTranslatef(1.5,0,0);          // transf. modelare
                        glutSolidSphere(1,16,16);
                glPopMatrix();
                glPushMatrix();
                        glTranslatef(-1.5,0,0);         // transf. modelare
                        glutSolidSphere(1,16,16);
                glPopMatrix();
        glPopMatrix();
        glutSwapBuffers();
}
```



(a) χ=0 (b) χ=45°

(e) χ=180° (f) χ=225°

Fig. 7.14 Sursă de lumină fixă în direcţia (1.0, 1.0, 1.0, 0.0).

Prin mişcarea de rotaţie a punctului de observare, sferele sunt văzute iluminate în diferite situaţii: pentru χ=45° se poate observa partea cea mai luminată a sferelor; pentru χ=225° se observă partea cea mai puţin luminată a sferelor.

Surse de lumină care se deplasează o dată cu punctul de observare. Pentru a crea surse de lumină care se mişcă odată cu punctul de observare, trebuie ca poziţia sursei să fie setată înainte de transformarea de observare. Poziţia sau direcţia sursei dată prin funcţia `glLight#()` cu parametrul `GL_POSITION`, este transformată cu matricea curentă din stiva de modelare-vizualizare care conţine matricea identitate, deci localizarea sursei rămâne constantă faţă de punctul de observare. Acest mod de control al sursei este folosit în simulatoare de antrenament, pentru a simula lumina farurilor vihiculului. Dacă poziţia sursei este (0,0,0,1), atunci sursa este plasată chiar în punctul de observare.

ν **Exemplul 7.5**

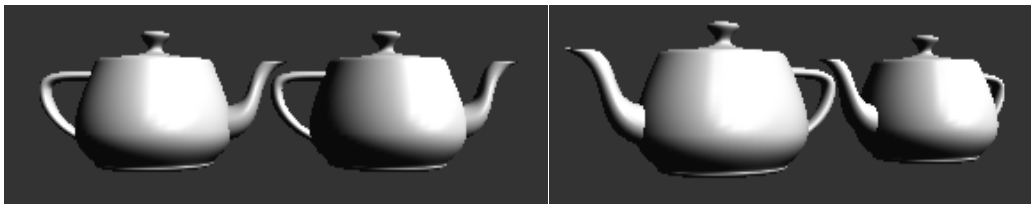
Se reia programul de generare a imaginii obiectelor luminate din exemplul precedent, dar sursa de lumină se setează cu o direcţie fixă în sistemul de referinţă de observare prin setarea direcţiei sursei înainte de transformarea de observare. În continuare sunt prezentate funcţiile callback `Display()` şi `Keyboard()` ale programului, celelalte funcţii fiind nemodificate (cu excepţia culorii de ştergere care este gri închis (0.3, 0.3, 0.3, 1.0) şi a înregistrării funcţiei callback `Keyboard()`).

```
static int gamma = 0;
void Keyboard(unsigned char key, int x, int y){
        switch (key){
        case 'G':
                gamma = (gamma+30)%360;
                glutPostRedisplay();
                break;
        case 'g':
                gamma = (gamma-30)%360;
                glutPostRedisplay();
                break;
        }
}
void Display(){
        GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};
        glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLightfv(GL_LIGHT0, GL_POSITION, light_position);
        glPushMatrix();
                glTranslatef(0,0,-8);
                glRotated(gamma,0,1,0);                 // transf. observare
                glPushMatrix();
                        glTranslatef(1.5,0,0);          // transf. modelare
                        glutSolidTeapot(1);
                glPopMatrix();
                glPushMatrix();
                        glTranslatef(-1.5,0,0);         // transf. modelare
                        glutSolidTeapot(1);
```

```
            glPopMatrix();
        glPopMatrix();
        glutSwapBuffers();
}
```

Sursa de lumină direcţională are direcţia (1.0, 1.0, 1.0, 1.0) care, transformată cu matricea identitate, păstrează aceeaşi valoare în sistemul de referinţă de observare. De aceea, indiferent de poziţia şi direcţia de observare, obiectele sunt iluminate dintr-o direcţie fixă faţă de direcţia de observare.

În Fig. 7.15 sunt reprezentate mai multe imagini, pentru direcţii de observare diferite. Unghiul χ (gamma în program) de rotaţie faţă de axa y se modifică prin comenzi de la tastatură, prelucrate de funcţia callback `Keyboard()`.



(a) χ=0 (b) χ = 30°

(c) χ=90° (c) χ=150°

Fig. 7.15 Sursă de lumină care se deplasează o dată cu punctul de observare.

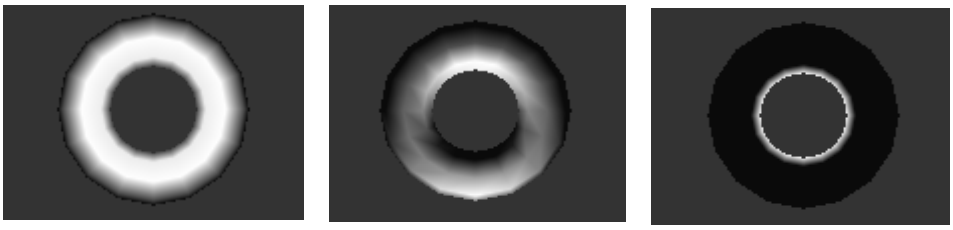
Surse de lumină care se deplasează în scenă. Se pot defini surse de lumină care se mişcă independent în scena virtuală, prin aplicarea unei transformări de instanţiere (care localizează sursa de lumină în sistemul de referinţă universal), urmată de transformnarea de observare, prin care se calculează poziţia sau direcţia sursei în sistemul de referinţă de observare. La fel ca şi în cazul instanţierii obiectelor în scenă, se foloseşte matricea curentă din stiva de modelare-vizualizare.

ν **Exemplul 7.6**

Se consideră un punct şi o direcţie de observare fixă, iar obiectele din scenă sunt luminate de o sursă de lumină direcţională care îşi modifică direcţia printr-o comandă de la mouse. Programul care implementează această funcţionare este similar celorlalte programe de iluminare prezentate în exemplele precedente, cu excepţia funcţiei `Display()` şi a funcţiei callback de prelucrare a evenimentelor de la mouse, care trebuie să fie înregistrată în `GLUT`.

```
static int teta = 0;
void Mouse(int button, int state, int x, int y){
        switch(button){
        case GLUT_LEFT_BUTTON:
                if (state == GLUT_DOWN){
                        teta = (teta+30)%360;
                        glutPostRedisplay();
                }
        }
}
void Display(){
        GLfloat light_position[] = {0, 0, 2, 1.0};
        glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        glPushMatrix();
                glTranslatef(0,0,-8);           // transf. observare
                glPushMatrix();
                        glRotated(teta,1,0,0);// transf. inst. sursa
                        glLightfv(GL_LIGHT0, GL_POSITION,
                                        light_position);
                glPopMatrix();
                glutSolidTorus(0.3, 0.8, 8, 16);
        glPopMatrix();
        glutSwapBuffers();
}
```

Iniţial variabila `teta = 0` şi sursa este plasată pe axa `z` la distanţă egală cu 2 faţă de centrul sistemului de referinţă universal. La fiecare apăsare a butonului stânga al mousului, variabila `teta` este incrementată (modulo 360) cu 30, şi poziţia sursei este rotită cu unghiul `spin` în raport cu axa x. Imaginile succesive (Fig. 7.16) indică modul în care este iluminat un tor de o sursă de lumină care se roteşte în jurul axei `x`.



(a) θ=0 (b) θ = 60° (c) θ = 180°

Fig. 7.16 Sursă de lumină care se deplasează în scenă.

Combinarea culorilor

Mai multe efecte în redarea obiectelor (transparenţă, anti-aliasing, ceaţă) se obţin prin combinarea culorilor la nivel de pixel.

Pentru validarea operaţiei de combinare a culorilor la nivel de pixel (*blending*) se apelează funcţia `glEnable(GL_BLEND)`. În cursul combinării, culoarea unui

nou fragment (sursa) este combinată cu culoarea memorată într-o locaţie corespunzătoare din bufferul de cadru, care este şi destinaţia pentru înscrierea rezultatului combinării.

Pentru combinarea culorilor se calculează factorii de combinare ai sursei şi ai destinaţiei. Aceşti factori sunt cvadrupleţi RGBA care se multiplică cu componentele corespunzătoare R, G, B şi A ale sursei, respectiv ale destinaţiei.

Fie factorii de combinare ai sursei (S_r, S_g, S_b, S_a) şi ai destinaţiei (D_r, D_g, D_b, D_a) şi componentele culorii sursei (R_s, G_s, B_s, A_s) şi ale destinaţiei (R_d, G_d, B_d, A_d). Culoarera rezultată prin combinare, care se înscrie în locaţia de destinaţie are componentele ($R_sS_r+R_dD_r$, $G_sS_g+G_dD_g$, $B_sS_b+G_dD_g$, $A_sS_a+A_dD_a$). Este posibil ca fiecare componentă a culorii rezultate să fie limitată la intervalul [0,1].

Factorii de combinare se pot selecta din mai multe valori posibile prin argumentele transmise funcţiei:

```
void glBlendFunc(GLenum sfactor, GLenum dfactor);
```

În tabelul 4.3 sunt date valorile constantelor simbolice predefinite în OpenGL, din care se pot selecta factorii de combinre ai sursei şi ai destinaţiei.

Tabelul 4.3 Factorii de combinare ai sursei şi ai destinaţiei

Constanta	**Sursă-destinaţie**	**Factorul de combinare**
GL_ZERO	sursă sau dest	(0,0,0,0)
GL_ONE	sursă sau dest	(1,1,1,1)
GL_DST_COLOR	sursă	(R_d,G_d,B_d,A_d)
GL_SRC_COLOR	destinaţie	(R_s,G_s,B_s,A_s)
GL_ONE_MINUS_DST_COLOR	sursă	(1,1,1,1)-(R_d,G_d,B_d,A_d)
GL_ONE_MINUS_SRC_COLOR	destinaţie	(1,1,1,1)-(R_s,G_s,B_s,A_s)
GL_SRC_ALPHA	sursă sau dest	(A_s,A_s,A_s,A_s)
GL_DST_ALPHA	sursă sau dest	(A_d,A_d,A_d,A_d)
GL_ONE_MINUS_SRC_ALPHA	sursă sau dest	(1,1,1,1)- (A_s,A_s,A_s,A_s)
GL_ONE_MINUS_DST_ALPHA	sursă sau dest	(1,1,1,1)- (A_d,A_d,A_d,A_d)
GL_SRC_ALPHA_SATURATE	sursă	(f,f,f,1); f=min(A_s,1-A_d)

Transparenţa

Transparenţa suprafeţelor se implementează folosind combinarea cu factorii de combinare `GL_SRC_ALPHA`, pentru sursă şi `GL_ONE_MINUS_SRC_ALPHA` pentru destinaţie.

- **Exemplul 7.7**

În programul următor se suprapun parţial suprafeţe transparente triunghiulare de culoare gri deschis (0.8, 0.8, 0.8) peste un dreptunghi de culoare gri închis (0.2, 0.2, 0.2). Imaginea din Fig. 7.17 s-a obţinut pentru valorile 0.25, 0.50, 0.75, 1.0 ale componentei A_s a triunghiului. Combinarea culorii unei noi suprafeţe cu culoarea existentă în buffer produce aspectul de transparenţă a suprafeţelor.

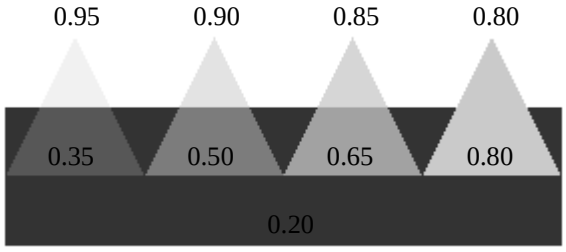


Fig. 7.17 Combinarea culorilor în calculul transparenţei.

În continuare sunt prezentate funcţiile callback `Init()` şi `Display()` ale programului dezvoltat sub `GLUT`. Celelalte funcţii, `main()` şi `Reshape()` nu au nimic special, fiind asemănătoare cu cele din programele precedente.

```
void Init(){
        glClearColor(1.0,1.0,1.0,1.0);
        glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
}
void Triangle() {
        glBegin(GL_POLYGON);
                glVertex3d(-2,-2,0);
                glVertex3d( 2,-2,0);
                glVertex3d( 0, 2,0);
        glEnd();
}
void Display(){
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glPushMatrix();
        glTranslated(0.0,0.0,-20.0);// transf. de observare

        glDisable(GL_BLEND);       // invalid. comb. culorilor
        glColor3d(0.2,0.2,0.2);
        glBegin(GL_POLYGON);    // desenare dreptunghi
                glVertex3d(-8,0,0);
                glVertex3d( 8,0,0);
                glVertex3d( 8,4,0);
                glVertex3d(-8,4,0);
        glEnd();

        glEnable (GL_BLEND);                    // validare comb. culorilor

        glPushMatrix();
                glTranslated(-6.0,4.0,0.0);
                glColor4d(0.8,0.8,0.8,0.25);
```

```
            Triangle();                         // al doilea triunghi
        glPopMatrix();

        glPushMatrix();
            glTranslated(-2.0,4.0,0.0);
            glColor4d(0.8,0.8,0.8,0.50);
            Triangle();                         // al doilea triunghi
        glPopMatrix();

        glPushMatrix();
            glTranslated(2.0,4.0,0.0);
            glColor4d(0.8,0.8,0.8,0.75);
            Triangle();                         // al treilea triunghi
        glPopMatrix();

        glPushMatrix();
            glTranslated(6.0,4.0,0.0);
            glColor4d(0.8,0.8,0.8,1.0);
            Triangle();         // al patrulea triunghi
        glPopMatrix();

        glPopMatrix();
        glutSwapBuffers();
}
```

În funcţia `Init()` se setează culoarea de ştergere albă (1.0, 1.0, 1.0, 1.0) a bufferului de culoare şi factorii de combinare `GL_SRC_ALPHA`, pentru sursă şi `GL_ONE_MINUS_SRC_ALPHA` pentru destinaţie.

Dreptunghiul se înscrie în buffer cu culorea lui iniţială. Culoarea fiecăruia din cele 4 triunghiuri care urmează (poligoane sursă) se combină la nivel de pixel cu culoarea existentă în buffer (care poate fi culoarea de ştergere sau culoarea dreptunghiului) folosind pentru factorii de combinare valoarea transparenţei sale. Primele trei triunghiuri sunt compuse din două culori: una rezultată din combinarea culorii proprii cu culoarea dreptunghiului, cealaltă rezultată din combinarea culorii proprii cu culoarea de ştergere. De exemplu, pentru primul triunghi, în partea de jos, suprapusă peste dreptunghi, fiecare componentă de culoare R,G,B are intensitatea 0.25 * 0.8 + (1- 0.25)*0.2 = 0.35; în partea de sus, prin combinare cu culoarea de ştergere, fiecare componentă de culoare R,G,B rezultată are intensitatea 0.25 * 0.8 + (1- 0.25)*1.8 = 0.95. Pentru al doilea triunghi componentele de culoare sunt 0.5 pentru partea de jos şi 0.9 pentru partea de sus; pentru al treilea triunghi componentele de culoare sunt 0.65 pentru partea de jos şi 0.85 pentru partea de sus. Ultimul triunghi, cu $A_s = 1$, înlocuieşte complet valorile existente în buffer şi este desenat cu componentele de culoare 0.8 în ambele părţi de suprapunere.

De remarcat că în acest program simplu nu este validat bufferul de adâncime. În mod implicit, în OpenGL testul de adâncime este invalidat şi, deoarece nu se validează nici la iniţializare nici în altă parte în program, testul de adâncime nu se efectuează şi poligoanele sunt înscrise în bufferul de imagine în ordinea în care sunt transmise, indiferent de adâncimea lor.

Acest lucru trebuie să fie evitat atunci când sunt reprezentate obiecte tridimensionale opace şi transparente în scenă. Dat fiind că obiectele tridimensionale se pot ascunde unele pe altele este necesar să se valideze testul de adâncime prin

apelul funcţiei `glEnable(GL_DEPTH_TEST)`. Obiectele opace se desenează mai întâi şi se execută operaţiile cu bufferul de adâncime în mod normal. Pentru desenarea obiectelor transparente, se setează condiţia de acces numai pentru citire a bufferului de adâncime (read-only) prin apelul funcţiei `glDepthMask(GL_FALSE)`. Atunci când se desenează obiecte transparente, adâncimea lor se compară cu adâncimile memorate în Z-buffer, stabilite de obiectele opace, astfel că nu se desenează părţile ascunse (aflate în spatele) obiectelor opace. Dacă sunt mai aproape de punctul de observare, se desenează prin combinarea culorii, dar nu se elimină adâncimea obiectele opace din bufferul de adâncime, deoarece acesta este read-only.

Anti-aliasing

Biblioteca OpenGL pune la dispoziţie două modalităţi de redare antialiasing: prin combinarea culorilor (*blending*) şi prin buffer de acumulare (*A-buffer*).

Redarea anti-aliasing prin combinarea culorilor este o tehnică de prefiltrare a imaginii şi dă rezultate de netezire numai dacă se renunţă la mecanismul de Z-buffer (buffer de adâncime) şi se desenează suprafeţele în ordinea în care sunt transmise bibliotecii. În această situaţie, culoarea unui pixel acoperit parţial de un poligon se obţine prin combinarea culorii poligonului cu culoarea pixelului, aflată în bufferul de culoare. În modul RGBA, OpenGL multiplică componenta A a culorii poligonului cu ponderea de acoperire (raportul dintre aria acoperită de fragmentul poligonului şi aria pixelului). Această valoare poate fi folosită pentru ponderarea culorii în fiecare pixel prin combinare cu factorul `GL_SRC_ALPHA` pentru sursă şi factorul `GL_ONE_MINUS_SRC_ALPHA` şi pentru destinaţie. Pentru efectuarea acestor calcule, mai este nesesară validarea anti-aliasing-ului prin apelul funcţiei `glEnable()` cu unul din argumentele `GL_POINT_SMOOTH`, `GL_LINE_SMOOTH`, `GL_POLYGON_SMOOTH`, pentru puncte, linii şi, respectiv, poligoane.

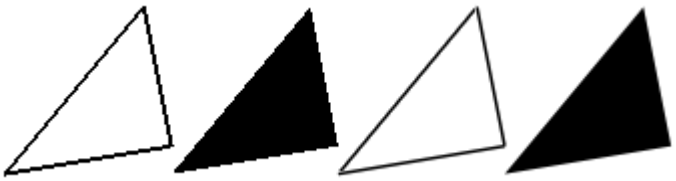
ν Exemplul 7.8

Programul din acest exemplu este cel folosit pentru generarea imaginilor a două triunghiuri cu şi fără antialiasing. Funcţiile callback `Init()` şi `Display()` ale programului dezvoltat sub `GLUT` sunt următoarele:

```
void Init(){
        glClearColor(1.0,1.0,1.0,1.0);
        glLineWidth (1.5);
  glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);        }
void Display(){
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glColor3f(0.0,0.0,0.0);

        glPushMatrix();
        glTranslated(0.0,0.0,-25.0);

        // Triunghiuri fără antialiasing
        glDisable(GL_BLEND);
        glDisable(GL_LINE_SMOOTH);
  glDisable(GL_POLYGON_SMOOTH);
```

```
        glPushMatrix();
        glTranslated(-8,0,0.0);
        glBegin(GL_LINE_LOOP);
                glVertex3d(-3,-3,0);
                glVertex3d( 3,-2,0);
                glVertex3d( 2,3,0);
        glEnd();
        glPopMatrix();
        glPushMatrix();
        glTranslated(-2,0,0.0);
        glBegin(GL_POLYGON);
                glVertex3d(-3,-3,0);
                glVertex3d( 3,-2,0);
                glVertex3d( 2,3,0);
        glEnd();
        glPopMatrix();

        // Triunghiuri cu antialiasing
        glEnable(GL_BLEND);
        glEnable(GL_LINE_SMOOTH);
  glEnable(GL_POLYGON_SMOOTH);

        glPushMatrix();
        glTranslated(4,0,0.0);
        glBegin(GL_LINE_LOOP);
                glVertex3d(-3,-3,0);
                glVertex3d( 3,-2,0);
                glVertex3d( 2,3,0);
        glEnd();
        glPopMatrix();

        glPushMatrix();
        glTranslated(10,0,0.0);
        glBegin(GL_POLYGON);
                glVertex3d(-3,-3,0);
                glVertex3d( 3,-2,0);
                glVertex3d( 2,3,0);
        glEnd();
        glPopMatrix();

        glPopMatrix();
        glutSwapBuffers();
}
```

Deoarece anti-aliasing-ul se realizează prin combinarea culorilor, este necesară şi validarea combinării culorilor (`glEnable(GL_BLEND)`) pentru poligoanele cu antialiasing. La execuţia acestui program se obţine imaginea din Fig. 7. 18.



(a) (b)

Fig. 7.18 (a) Două poligoane fără anti-aliasing
(b) Aceleaşi poligoane reprezentate cu anti-aliasing.

Acest mod de realizare a filtrării anti-aliasing este foarte simplist şi nu poate fi folosit decât pentru suprafeţe sau linii care nu se suprapun şi pot fi redate fără eliminarea suprafeţelor ascunse, aşa cum a fost situaţia din acest exemplu.

Dacă se invalidează bufferul de adâncime, eliminarea suprafeţelor ascunse trebuie să fie asigurată printr-un alt mecanism, de exemplu prin ordonarea suprafeţelor după adâncime (lista de display). Dar eliminarea suprafeţelor ascunse prin ordonarea suprafeţelor după adâncime este un procedeu costisitor şi care nu poate fi implementat pentru scene complexe, cu un număr mare de obiecte. Soluţia acceptabilă ca performanţe şi rezultate pentru redarea anti-aliasing a scenelor complexe este folosirea unui buffer de acumulare (A-buffer). Implementarea unui A-buffer prin rutine software este foarte lentă şi deci impracticabilă. Sistemele grafice performante implementează diferite tehnici de anti-aliasing în hardware. Aceste tehnici sunt dependente de implementarea hardware şi sunt mai greu de descris la modul general.

8

Crearea şi vizualizarea scenelor virtuale

Modul în care se creează şi se utilitează modelele obiectelor tridimensionale variază în funcţie de aplicaţia grafică căreia îi sunt destinate.

În aplicaţii de proiectare grafică pentru diferite domenii de activitate (mecanică, arhitectură), crearea şi utilizarea modelelor sunt tratate împreună, în cadrul unui sistem de proiectare, cum este, de exemplu sistemul AutoCad. Obiectele create de proiectant sunt redate pe display, memorate într-o reprezentare specifică sistemului şi editate sau modificate în continuare în cursul proiectării.

Modelele din care este compusă scena într-o aplicaţie de realitate virtuală au cerinţe diferite de timp de prelucrare în timpul creerii scenei (*modelare*) şi a utilizării ei (*redare*). În mod obişnuit cele două procese, de modelare şi de redare a modelelor se desfăşoară separat.

Modelarea este un proces off-line prin care se creează colecţia de modele care compun scena virtuală. Procesul de modelare a obiectelor se desfăşoară asistat de calculator, în diferite sisteme de proiectare interactivă, care pun la dispoziţia proiectantului funcţii de creare a modelelor pornind de la diferite date de intrare. Pentru aplicaţiile de realitate virtuală pot fi folosite atât sisteme de proiectare automată de uz general, cum sunt, de exemplu Autocad sau 3DStudio, sau sisteme de proiectare special concepute pentru crearea scenelor virtuale, numite sisteme de generare a bazelor de date (DBGS – *Data Base Generation Systems*). Astfel de sisteme, cum sunt sistemul Multigen, Corypheus, Sense8 şi multe altele, oferă numeroase posibilităţi de modelare a obiectelor tridimensionale, a terenului, şi a fenomenelor atmosferice (nori, ceaţă, etc.). Rezultatul operaţiei de modelare îl reprezintă modele ale obiectelor sau colecţii de modele care compun o scenă virtuală, şi acestea sunt memorate sub forma de *baze de date grafice*, alcătuite dintr-unul sau mai multe fişiere.

Redarea imaginii obiectelor este un proces care se desfăşoară interactiv, de cele mai multe ori în timp real, în cursul diferitelor experimente de realitate virtuală (simulatoare de antrenament, muzee virtuale, etc.), într-un sistem grafic numit generator de imagine (*Image Generator* –IG). În generatorul de imagine se re-creează modelele obiectelor şi al scenei virtuale, folosind ca date de intrare fişiere de baze de date. Pentru redarea eficientă a imaginii, scena virtuală este organizată ierarhic, pe mai multe nivele de ierarhie, ceea ce permite selectarea rapidă a obiectelor, moşteniri de transformări geometrice şi de atribute, etc. Această organizare ierarhică se implementează printr-un graf aciclic direcţionat, numit *graful scenei* (*scene graph*). Nodurile grafului conţin modele ale obiectelor sau informaţii de redare (gruparea obiectelor în structura ierarhică, poziţia obiectelor, proprietăţi de redare, lumini, etc.). Organizarea ierarhică a scenei virtuale va fi prezentată mai detaliat în secţiunea următoare.

Bazele de date grafice sunt reprezentarea pe suport de memorie (fişiere) a modelelor obiectelor scenei şi trebuie să conţină toate informaţiile necesare pentru crearea grafului scenei. Există un număr mare de formate de reprezentare a bazelor de date grafice, mai mult sau mai puţin complete din punct de vedere al informaţiilor de construire a grafului scenei virtuale. Proliferearea formatelor bazelor de date îngreunează portabilitatea programelor de aplicaţii grafice, care trebuie să prevadă posibilităţi de interpretare pentru cât mai multe formate.

În orice generator de imagine, modelul scenei are o structură bine precizată, care depinde de scopul aplicaţiei, iar datele importate din diferite formate de baze de date sunt convertite la formatul intern al scenei. Un tip de format de bază de date reprezintă un anumit mod de descriere a entităţilor componente ale bazei de date. Cele mai cunoscute dintre formatele de baze de date sunt date în Tabelul 8.1.

Tabelul 8.1 Formate de baze de date.

Denumire	**Descriere**
3ds	Format binar 3DStudio, firma AutoDesk
dxf	Format ASCII AutoCad, firma AutoDesk
obj	Format ASCII, firma Wavefront Technologies
phd	Format de descriere poliedre, firma Silicon Graphics
iv	Format ASCII Open Inventor, firma Silicon Grafics
dwb	Format binar/ASCII, firma Corypheus
flt14	Format binar Open Flight, firma Multigen

Formatele `3ds` şi `dxf` sunt formate de export ale sistemelor de proiectare de uz general 3DStudio şi AutoCad de la firma AutoDesk. Ele nu au fost concepute special pentru propiectarea bazelor de date grafice, astfel că nu pot avea decât o utilizare limitată. De exemplu, se pot folosi pentru proiectarea de obiecte individuale, care sunt apoi înglobate în scene decrise prin formate mai complete.

Formatul `obj` este un format simplu care poate descrie obiecte tridimensionale reprezentate prin reţea de poligoane. Firma Wavefront Technology a plasat în domeniu public un număr de modele în format `obj`, care a devenit în felul acesta destul de cunoscut. O bogată librărie de modele produse de firma Viewpoint (peste 3500 de modele tridimensionale reprezentând avioane, maşini, animale, organe anatomice, etc.) sunt disponibile în formate `dxf`, `obj`, `flt`. Modelul avionului F-16 din Fig. 2.16 este un model în formatul `obj`.

Formatele `dwb` şi `flt` sunt formate speciale de descriere a bazelor de date grafice, dezvoltate împreună cu sistemele de generare a bazelor de date de companiile Corypheus şi, respectiv, Multigen. Aceste formate conţin informaţii complete privind obiectele, amplasarea şi gruparea lor, materiale, lumini, din care se poate crea graful scenei în generatorul de imagine.

Formatul `iv` (Open Inventor), dezvoltat de firma Silicon Graphics, este un superset al limbajului de modelare VRML 1.0 (*Virtual Reality Moddeling Language*), care în 1997 a devenit standard sub numele de VRML 97. Limbajul VRML 97 permite crearea scenelor virtuale şi accesarea lor prin internet. O scenă virtuală în limbajul VRML este un format de bază de date, asemănător altor formate de baze de date. Denumirea de limbaj VRML provine din faptul că specificaţiile VRML conţin reguli sintactice şi semantice precise, care permit verificarea corectitudinii construcţiilor folosite. Acest limbaj este descris în secţiunea următoare.

8.1 Reprezentarea scenelor virtuale

O scenă virtuală (*virtual scene*) este compusă dintr-o colecţie de obiecte tridimensionale, specificate prin forma şi poziţia lor, prin aspect (culoare, material, etc), şi comportare (deplasare în spaţiu, interacţiune). Diferite alte denumiri mai sunt utilizate cu acelaşi înţeles: lume virtuală *(virtual world*), mediu virtual (*virtual environment*). Construirea ierarhică a scenei permite organizarea şi parcurgerea eficientă a acesteia. Scena este compusă din noduri conectate prin arcuri direcţionate. Un nod în scenă descrie o anumită entitate: un obiect tridimensional, o grupare de obiecte tridimensionale, o transformare geometrică, o textură, o sursă de lumină, etc. Un arc al grafului scenei introduce un descendent (fiu) al unui nod, care moşteneşte unele din atributele nodului părinte (Fig. 8.1).

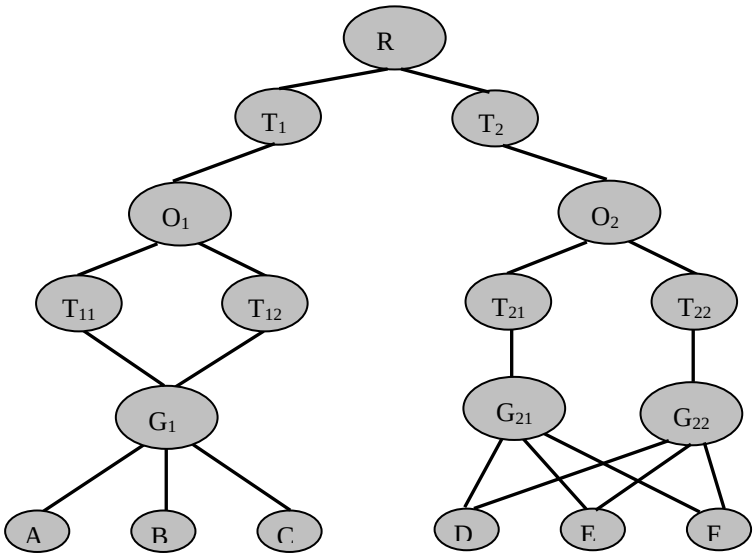


Fig. 8.1 Graful unui scene virtuale reprezentat printr-o ierarhie de noduri.

Construirea ierarhică a scenei permite reutilizarea obiectelor şi a transformărilor geometrice. Un obiect complex se compune din gruparea mai multor obiecte mai simple, dintre care unele se repetă, în poziţii de instanţiere diferite. În felul acesta, un nod poate avea mai mult de un nod părinte.

Nodurile grafului sunt de mai multe tipuri, dintre care cele mai importante sunt: nodurile de descriere geometrică (forma) a obiectelor tridimensionale, noduri de transformare şi noduri de grupare.

Nodurile de descriere geometrică conţin modele ale unor obiecte elementare, reprezentate poligonal, parametric sau prin descrierea matematică într-un sistem de referinţă local (de modelare). În Fig.8.1 nodurile A, B, C, D, E, F reprezintă obiecte tridimensionale. Nodul G_1 grupează mai multe obiecte simple, formând un obiect compus care, la rândul lui este instanţiat de două ori, în poziţii definite prin intermediul nodurilor de transformare T_1 şi T_2, rezultând obiecul O_1, s.a.m.d. Nodul R este un nod grup şi reprezintă rădăcina grafului scenei.

Graful scenei descrie relaţiile de moştenire între noduri. Atributele unui obiect descris de un nod pot fi cele moştenite de la nodul părinte, pot fi completate, sau redefinite în nodul fiu.

Scena virtuală este definită în sistemul de referinţă universal, în care se poziţionează şi orientează punctul de observare în fiecare cadru al imaginii generate. Obiectele din nodurile frunză sunt reprezentate într-un sistem de referinţă local. Un nod de transformare conţine o matrice de transformare **M** care defineşte poziţia şi orientarea sistemului de referinţă al nodului fiu în sistemul de referinţă al nodului părinte. Transformarea prin aplicarea matricei **M** punctelor obiectelor din nodul fiu, le transformă din sistemul de referinţă al nodului fiu în sistemul de referinţă al nodului părinte.

Redarea imaginii unei scene reprezentată printr-un graf aciclic direcţionat se obţine prin parcurgerea (*traversarea*) grafului scenei. Traversarea grafului scenei este o operaţie de parcurgere a grafului în adâncime (depth first search), care începe cu nodul rădăcină al grafului şi parcurge toate drumurile în graf până la fiecare nod frunză. În acest fel, un nod poate fi traversat de mai multe ori, pe căi diferite de parcurgere. De exemplu, în graful din Fig. 8.1 nodul A este instanţiat de două ori, odată pe calea $RT_1O_1T_{11}G_1A$ şi a doua oară pe calea $RT_1O_1T_{12}G_1A$; la fel şi nodurile B şi C.

În reprezentarea ierarhică a obiectelor şi a scenei, un obiect tridimensional este instanţiat (plasat în sistemul de referinţă universal) prin aplicarea unei secvenţe de transformări de instanţiere, şi anume toate transformările din nodurile de transformare de la rădăcină până la nodul frunză pe una din căile de parcurgere a grafului. Combinarea dintre operaţia de parcurgere în adâncime a grafului şi operaţiile de concatenare a matricelor de transformare prin intermediul stivelor de transformare (definite în OpenGL sau de programul de aplicaţie) permite reutilizarea matricelor

calculate pe parcursul traversării prin salvarea lor în stivele de transformări şi restaurarea atunci când sunt necesare.

Algoritmul de traversare a grafului scenei de mai jos (în pseudocod asemănător limbajului C) specifică parcurgerea în adâncime a grafului scenei şi operaţiile care se execută la vizitarea fiecărui nod, în funcţie de tipul acestuia. Se consideră un tip de date `Node`, care poate fi o clasă de obiecte (în programarea orientată pe obiecte) sau o structură (în abordarea procedurală) şi o stivă de matrice de transformare de modelare, care admite operaţiile de introducere (`PushMatrix()`), de extragere (`PopMatrix()`) şi de concatenare (prin postmultiplicare-`MultMatrix()`) a matricei curente de transformare **C** cu o nouă matrice **M**, cu depunerea rezultatului în matricea curentă (**C** = **C M**). În mod obişnuit, aceasta este stiva OpenGL de transformare de modelare-vizualizare (stiva `MODELVIEW`). În această prezentare se consideră trei tipuri de noduri: noduri frunză cu descrierea formei unui obirect (`SHAPE`), noduri grup (`GRUP`) şi noduri de transformare (`TRANSFORM`).

```
void Traversal (Node* node){
        if (node->type == SHAPE){
                Redare obiect folosind stiva curenta C
        }
        else if (node->type ==GROUP){
                for (i=0;i< nChilds; i++){
                        PushMatrix();
                        Traversal(node->child[i]);
                        PopMatrix();
                }
        }
        else if (node->type == TRANSFORM){
                MultMatrix(M);
        }
}
```

Notaţiile care s-au folosit în acest algoritm sunt evidente. Tipul nodului (`node->type`) determină modul de prelucrare a acestuia. Un nod grup are `nChilds` noduri fii, numerotaţi `childs[i]`. Un nod de transformare conţine o matrice de transformare **M**. Travesarea începe prin iniţializarea matricei curente de transformare cu matricea identitate (**C** = **I**) şi apelul funcţiei de traversare pentru nodul rădăcină al scenei (`Traversal(R)`).

Matricele de instanţiere se introduc în stivă în ordinea inversă aplicării lor asupra obiectului, aşa cum este necesar în operaţia de acumulare prin postmultiplicare. De exemplu, pentru prima instanţă a obiectului A, se execută operaţiile:

$\mathbf{C} = \mathbf{I}$
$\mathbf{C} = \mathbf{C}\,\mathbf{M}_1 = \mathbf{M}_1$
$\mathbf{C} = \mathbf{C}\,\mathbf{M}_{12} = \mathbf{M}_1\mathbf{M}_{12}$

Un punct oarecare P a obiectului A este instanţiat prin transformarea:

$\mathbf{P'} = \mathbf{CP} = \mathbf{M}_1\mathbf{M}_{12}\mathbf{P}$

Pentru o altă instanţiere a nodului A, matricea intermediară $\mathbf{M_1}$ este salvată înainte de prelucrarea fiecăruia din fiii nodului O_1. Poate este mai puţin evidentă eficienţa metodei în acest exemplu foarte mic, dar pentru scene complexe, cu multe transformări succesive, economia de calcule de produse de matrice este considerabilă.

Acesta este modul principial de organizare şi traversare a grafului unei scene virtuale. Construirea unui astfel de graf se poate face fie direct, prin program (pentru mici exemple cu scene de dimensiuni reduse) sau, în forma cea mai generală, prin parcurgerea unei baze de date descrise într-un limbaj (format) care să permită identificarea construcţiei ierarhice a scenei: tipuri de noduri, atributele acestora, etc., pentru construcţia grafului echivalent. Un astfel de limbaj este limbajul VRML.

8.2 Limbajul VRML

Limbajul *VRML (Virtual Reality Modeling Language*) este un limbaj folosit în crearea scenelor virtuale tridimensionale, cu posibilitatea de acces a acestora prin reţeaua internet WWW (*World Wide Web*). Versiunea folosită în momentul de faţă este VRML 2.0 (numită şi VRML 97, fiind standard ISO din anul 1997) şi este o revizuire a versiunii VRML 1.0, stabilită în 1995. Numeroase informaţii referitoare la limbajul VRML se pot găsi în reţeaua internet (`http://www.vrml.org`).

În prima versiune, limbajul VRML 1.0 permitea crearea unei scene virtuale compusă din obiecte tridimensionale, dar cu posibilităţi limitate de interactivitate prin reţeaua internet. VRML 1.0 s fost dezvoltat pe baza formatului de descriere a bazelor de date Open Inventor, dezvoltat de firma Silicon Graphics, care suportă descrierea scenelor ierarhice cu obiecte poligonale, lumini, materiale şi efecte realiste. Un subset a formatului Open Inventor, cu extensii pentru acces prin reţeaua internet, a constituit baza de pornire pentru limbajul de descriere a scenelor virtuale VRML. În momentul de faţă, numeroase titluri de cărţi, articole, reviste, documentaţie, exemple, adrese de grupuri de lucru, etc., se găsesc pe reţea despre limbajul VRML şi utilizarea acestuia.

Un fişier VRML este un fişier ASCII care conţine descrierea unei scenei virtuale şi a unor acţiuni interactive, şi poate fi accesat prin reţea folosind programe de navigare (browsere). Pentru a afişa o scenă VRML din reţeaua internet sau dintr-un fişier de pe disc (cu extensia .vrl) este necesar un browser VRML, configurat ca o extensie (plug-in) al unui browser Web. Pentru browserul Netscape Navigator, există browserul Cosmo Player (de la compania Platinum Technology) pentru interpretarea fişierelor VRML. Un browser VRML citeşte fişierul care conţine descrierea scenei virtuale, crează graful scenei şi memorează datele de comportare a obiectelor şi de interacţiuni corespunzătoare. Observarea scenei virtuale şi navigarea în direcţia dorită se realizează prin comenzi ale browserului. Aceste comenzi sunt foarte simple şi se găsesc în Help.

Deşi utilizarea cea mai frecventă a limbajului VRML este în crearea şi redarea scenelor 3D prin reţeaua www, un fişier VRML poate fi folosit şi ca descriere a unei scene virtuale pentru aplicaţii de realitate virtuală care nu se desfăşoară în reţea. În această situaţie, pentru citirea fişierelor VRML se foloseşte un program de interpretare a fişierelor VRML (parser) care creează graful scenei în concordanţă cu structura programului de aplicaţie. În domeniul public pe reţeaua internet sunt disponibile un

mare număr de parsere care pot fi integrate într-o aplicaţie. De exemplu, există VRML 2.0 Parser de la Silicon Graphics (`http://vrml.sgi.com/`) sau Viper Parser (de la *NIST- National Institute of Standard and Technology*) implementat în limbajul Java, orientat pe obiecte.

În oricare situaţie, formatul VRML permite construirea grafului scenei, furnizând toate informaţiile despre obiectele tridimensionale, ierarhia acestora, condiţii de mediu ambiant, materiale, etc. Graful scenei, construit prin citirea unuia sau mai multor fişiere VRML, este apoi traversat pentru obţinerea imaginii obiectelor în diferite situaţii de navigare a scenei. Redarea imaginii obiectelor pe display se realizează prin intermediul bibliotecilor grafice de nivel mai scăzut, cum sunt OpenGL sau Direct3D (de la Microsoft). Aceste biblioteci asigură interfaţa corespunzătoare cu acceleratorul grafic al sistemului.

Specificaţiile limbajului VRML 97 descriu conceptele de realizare a scenelor virtuale, descrierea nodurilor, descrierea câmpurilor, a evenimetelor şi interacţiunea între noduri. Similar limbajelor de programare standard, specificaţiile VRML conţin reguli sintactice şi reguli semantice ale limbajului.

Un fişier VRML conţine un header (acelaşi pentru ISO VRML 97 şi VRML 2.0), urmat de descrierea nodurilor grafului scenei, care specifică formele geometrice, grupurile, transformările şi atributele mediului virtual care se contruieşte şi vizualizează. Sintaxa VRML este simplă, cu cuvinte cheie în limba Engleză sugestive pentru entitatea definită. Headerul fişierului este:

```
#VRML 2.0 utf8
# Alte comentarii
```

unde `utf8` este setul de caractere folosit. Este obligatorie introducerea versiunii `VRML 2.0` imediat după caracterul # (fără nici un spaţiu liber).

Un nod este specificat prin tipul nodului, urmat de un bloc cuprins între acolade. Blocul din interiorul acoladelor conţine zero sau mai multe câmpuri (*fields*) şi pentru fiecare câmp se specifică valoarea (sau valorile) acestuia (*field value*). Fiecare câmp al unui nod are un nume şi admite una sau mai multe valori de un tip de date determinat de tipul parametrului.

În VRML este extinsă noţiunea de nod pentru orice entitate care grupează mai multe informaţii. Se poate observa diferenţa dintre noţiunea de *nod VRML* şi noţiunea de nod într-un graf al scenei. Un nod într-un graf este o componentă a unei structuri ierarhice, fiecare nod având unul sau mai mulţi părinţi (cu excepţia nodului rădăcină care nu are nici un părinte) şi unul sau mai mulţi fii (cu excepţia nodurilor frunză care nu au fii). Un nod VRML poate fi nod în graful scenei, sau poate să fie inclus (ca valoare a unui câmp) într-un alt nod. Modul cum se construieşte graful scenei pornind de la specificarea nodurilor VRML depinde de programul de aplicaţie.

Sistemele de coordonate folosite în VRML sunt aceleaşi cu sistemle de coordonate din OpenGL: sistemul de referinţă universal este un sistem drept, sistemul de referinţă observator este de asemenea un sistem de referinţă drept, definit ca localizare şi orientare relativ la sistemul de referinţă universal; proiecţia perspectivă se defineşte printr-un plan perpendicular pe axa z a sistemului de referinţă de observare, cu direcţia de observare spre –z .

Tipurile de noduri VRML sunt următoarele:

- Forme şi descrieri geometrice (*shapes*): `Box`, `Cone`, `Coordinate`, `Cylinder`, `EvaluationGrid`, `Extrusion`, `IndexedFaceSet`, `IndexedLineSet`, `Normal`, `PointSet`, `Shape`, `Sphere`, `Text`.
- Aspect (*appearance*): `Appearance`, `Color`, `FontStyle`, `ImageTexture`, `Material`, `MovieTexture`, `PixelTexture`, `Texture-Coordinate`, `TextureTransform`.
- Grupuri (*grouping*): `Anchor`, `Billboard`, `Collision`, `Group`, `Inline`, `LOD`, `Switch`, `Transform`.
- Mediu (*environment*): `AudioClip`, `Background`, `Directional-Light`, `Fog`, `PointLight`, `Sound`, `SpotLight`.
- Vizualizare (*viewing*): `NavigationInfo`, `Viewpoint`.
- Animaţie (*animation*): `ColorInterpolator`, `Coordinate-Interpolator`, `NormalInterpolator`, `Orientation-Interpolator`, `PositionInterpolator`, `ScalarInterpolator`, `TimeSensor`.
- Interacţiuni (*interaction*): `CylinderSensor`, `PlaneSensor`, `ProximitySensor`, `SphereSensor`, `TouchSensor`, `VisibilitySensor`.
- Alte noduri: `Script`, `WordInfo`.

În total sunt 54 de tipuri de noduri, şi fiecare tip admite unul sau mai multe tipuri de câmpuri. Tipurile de date prin care se specifică valoarea unui câmp pot fi: numere întregi, numere în virgulă mobilă, valori boolene, şiruri de caractere, sau pot fi folosite chiar noduri VRML. Un exemplu de fişier VRML foarte simplu este dat în continuare.

ν **Exemplul 8.1**

```
#VRML 2.0 utf8
# Construieste un cilindru 3D
Shape {
        appearance Appearance {
                material Material {
                        diffuseColor 0.0, 0.5, 1.0
                }
        }
        geometry Cylinder {
                height 2.0
                radius 1.5
        }
}
```

Nodul de tipul `Shape` introduce un obiect tridimensional pentru care se specifică câmpurile cu numele `appearance` şi `geometry`, pentru definirea aspectului şi a formei. Valoarea câmpului `appearance` este un nod de tipul `Appearance`, definit imediat prin blocul corespunzător între acolade. Nodul

Apperence are specificat câmpul material a cărui valoare este dată printr-un nod de tipul Material. Nodul de tip Material conţine câmpul diffuseColor, pentru care se specifică valoarea prin trei numere în virgulă flotantă, pentru componentele roşu, verde, albastru. Valoarea câmpului geometry este un nod de tipul Cylinder, care este definit prin câmpurile heigth şi radius, cu valori numere în virgulă mobilă.

ν

Un fişier VRML poate conţine mii de noduri şi fiecare nod conţine un număr variabil de câmpuri, tipic două-trei câmpuri, dar pot fi şi zeci de câmpuri. Din fericire, toate câmpurile nodurilor au prevăzute valori implicite rezonabile, astfel că nodurile pot fi folosite chiar dacă nu se specifică valoarea tuturor câmpuri. De exemplu, valoarea implicită a culorii de ştergere este negru, şi, dacă nu se specifică altă culoare de fundal (printr-un nod Background), atunci, la navigare prin intermediul browserelor fundalul rămâne negru. Poziţia iniţială a punctului de observare este este la (0.0, 0.0, 10.0) şi, dacă nu se specifică altă poziţie, (printr-un nod Viewpoint), atunci navigarea începe cu această poziţie.

Sunt considerate comentarii toate caracterele după caracterul # până la sfârşitul liniei. Nodurile, câmpurile şi valorile câmpurilor pot fi aranjate în oricâte linii; spaţiile, tab-urile, virgulele şi caracterul de linie nouă sunt considerate spaţii albe (*white spaces*) şi ignorate la citire. VRML este case-senzitive, adică diferenţiază literele mici de cele mari.

8.2.1 Construirea formelor geometrice

Orice formă geometrică se construieşte folosind un nod de tipul Shape, cu câmpurile appearance şi geometry pentru specificarea aspectului şi a formei geometrice.

Forma geometrică se specifică prin numele unuia din corpurile geometrice elementare recunoscute de VRML (Box, Cylinder, Cone, Sphere), printr-o grilă de valori de altitudini (ElevationGrid), printr-un obiect obţinut prin extrudare (Extrusion) sau printr-o reţea de feţe poligonale (IndexedFaceSet), linii (IndexedLineSet) sau puncte (PointSet).

Cea mai puternică şi mai folosită modalitate de reprezentare a obiectelor este reprezentarea poligonală descrisă de nodul IndexedFaceSet. Acest nod defineşte un obiect printr-o mulţime de vârfuri (coordonate 3D) şi o mulţime de feţe, fiecare faţă fiind o listă de indici în vectorul de vârfuri. Se poate urmări mai simplu definiţia unui astfel de nod în exemplul următor.

ν **Exemplul 8.2**

```
#VRML V2.0 utf8
# cub.wrl - construieşte un cub prin feţe indexate
Shape {
    appearance Appearance {
```

```
        material Material { }
    }
    geometry IndexedFaceSet {
        coord Coordinate {
            point [
            # Coordonatele de deasupra a cubului
                -1.0  1.0  1.0,
                 1.0  1.0  1.0,
                 1.0  1.0 -1.0,
                -1.0  1.0 -1.0,
            # Coordonatele de jos ale cubului
                -1.0 -1.0  1.0,
                 1.0 -1.0  1.0,
                 1.0 -1.0 -1.0,
                -1.0 -1.0 -1.0
            ]
        }
        coordIndex [
            0, 1, 2, 3, -1,
                        7, 6, 5, 4, -1,
                        0, 4, 5, 1, -1,
                        1, 5, 6, 2, -1,
                        2, 6, 7, 3, -1,
                        3, 7, 4, 0
        ]
    }
}
```

Acest nod creează un cub şi specifică fiecare faţă poligonală printr-o listă de indici în lista de vârfuri şi în lista de culori. Orice vector (vectorul `point`, vectorul `coordIndex`) este încadrat între paranteze drepte. Delimitarea între indecşii feţelor se face prin numărul –1. ν

8.2.2 Noduri de grupare

Nodurile de grupare (`Group`, `LOD`, `Switch`, `Billboard`, `Inline`, `Collision`, `Anchor`, `Transform`) ale limbajului VRML sunt noduri care au unul sau mai multe noduri fii.

Nodul `Group` colectează mai mulţi fii, care pot fi forme, lumini, sau alte noduri `Group`. În exemplul următor este creat un obiect compus din trei paralelipipede dreptunghice (semnul "plus" tridimensional).

ν Exemplul 8.3

```
#VRML V2.0 utf8
# semn3D.wrl – semnul "plus" tridimensional
Group {
  children [
                        Background {
      skyColor  1.0 1.0 1.0
        },
```

```
        Viewpoint {
      position 40.0, 10.0, 40.0
      orientation 0.0 1.0 0.0 0.78
            },
            DirectionalLight {
      direction  0.0 -1.0 -0.2
      intensity 1.0
      ambientIntensity 0.2
      color 1.0 1.0 1.0
    },
    Shape {
      appearance DEF White Appearance {
        material Material { }
      }
      geometry Box {
        size 25.0 2.0 2.0
      }
    },
          Shape {
               appearance USE White
               geometry Box {
                    size 2.0 25.0 2.0
               }
          },
          Shape {
               appearance USE White
               geometry Box {
                    size 2.0 2.0 25.0
               }
          }
     ]
}
```

Imaginea care se obţine la vizualizarea acestui fişier VRML este dată în Fig. 8.2 . Grupul conţine şase noduri fii, trei noduri de descriere a unei forme geometrice (nodurile `Shape`), un nod de sursă lumină (`DirectionalLight`), un nod de definire a culorii de ştergere (`Background`) şi un nod de poziţionare a punctului de observare (`Viewpoint`). Lista de fii este introdusă între paranteze drepte precedate de cuvântul cheie `children`.

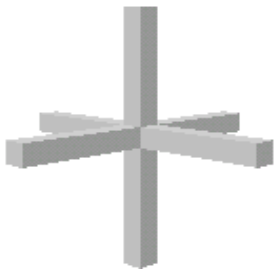


Fig. 8.2 Imaginea afişată de browserul Cosmo Player
la încărcarea fişierului semn3D.wrl.

În acest exemplu este folosită etichetarea nodurilor. Orice nod poate primi o etichetă introdusă prin cuvântul-cheie `DEF`. Un nod odată etichetat la definiţie, poate fi folosit ca valoare a unui câmp prin simpla introducere a cuvântului-cheie `USE` urmat de eticheta nodului. În exemplul de mai sus, un nod de aspect a fost etichetat cu numele `White` (`DEF White Appearance{.....}`) şi apoi folosit cu acest nume ca valoare a unui câmp (`appearance USE White`).

Această imagine se obţine fie prin intermediul unui browser (Cosmo Player pentru Netscape), fie prin interpretare (parsarea) fişierului, crearea grafului scenei şi redarea imaginii acestuia.

În fişierul `semn3D.wrl` mai este folosit un nod pentru definirea culorii de fundal (nodul `Background` cu câmpul `skyColor`), nodul `Viewpoint` pentru definirea poziţiei de observare şi nodul `DirectionalLight` pentru definirea unei surse de lumină direcţională. Se pot remarca unele diferenţe între convenţiile de definire a unor date în VRML şi cele din OpenGL. De exemplu, pentru rotaţie se specifică mai întâi componentele axei de rotaţie, apoi unghiul de rotaţie în radiani (în OpenGL se dă mai întâi valoarea unghiului în grade, apoi componentele axei de rotaţie).

Nodul `Switch` conţine o listă ordonată de noduri fii care conţin reprezentări variate ale aceluiaşi obiect. În cursul redării, se selectează în secvenţă câte unul din fii la fiecare cadru de imagine, ceea ce creează impresia de evoluţie (modificare dinamică) a obiectului. În acest fel sunt redate luminile de semnalizare (care îşi schimbă culoarea sau intensitatea la intervale stabilite de timp) sau flăcările, fumul, etc.

Nodul `LOD` conţine reprezentarea unui obiect cu nivele de detaliu multiple. Fiecare nod fiu al nodului `LOD` este selectat într-un interval de distanţe specificat în vectorul de distanţe al nodului `LOD`.

Nodul `Transform` defineşte transformarea de coordonate a tuturor nodurilor fii ai acestuia faţă de nodul părinte. Prin aplicarea acestei transformări, un nod fiu este transformat din sistemul de referinţă propriu în sistemul de referinţă al nodului părinte. Succesiunea de transformări care se aplică unui nod frunză îl transformă din sistemul de referinţă local în sistemul de referinţă universal.

ν Exemplul 8.4

Fişierul `cupola.vrl` de mai jos construieşte o cupolă din două obiecte poziţionate folosind un nod `Transform`:

```
#VRML V2.0 utf8
# cupola.wrl - construieşte o cupolă
Group {
  children [
        Background {
      skyColor  1.0 1.0 1.0
        },
                        DirectionalLight {
```

```
        direction  0.0 -1.0 -1.0
        intensity 1.0
        ambientIntensity 0.2
        color 1.0 1.0 1.0
          },
                        # Pereţii cupolei
          Shape {
                        appearance DEF Brown Appearance {
       material Material {
          diffuseColor 0.6 0.4 0.0
       }
    }
                        geometry Cylinder {
                    height 2.0
                    radius 2.0
                }
            },
            # Acoperişul cupolei
            Transform {
                translation 0.0 2.0 0.0
                children [
                    Shape {
                        appearance USE Brown
                        geometry Cone {
                            height 2.0
                            bottomRadius 2.5
                        }
                    }
                ]
            }
        ]
}
```

Cupola este reprezentată printr-un nod `Group` cu doi fii. Primul fiu este un cilindru, care redă pereţii cupolei; cel de-al doilea fiu este un con translatat cu 2 unităţi faţă de nodul părinte (nodul `Group`) şi reprezintă acoperişul. Imaginea obţinută prin încărcarea fişierului `cupola.wrl` este dată în Fig. 8.3. (a).

Fig. 8.3 (a) Imaginea fişierului cupola.wrl.
(b) Imaginea fişierului asterix.wrl

ν Exemplul 8.5

```
#VRML V2.0 utf8
# asterix.wrl – construieşte semnul “asterix” 3D
Group {
   children [
                                    Viewpoint {
         position 40.0, 10.0, 40.0
         orientation 0.0 1.0 0.0 0.78
                  },
   # Primul braţ
      DEF Arm1 Shape {
         appearance Appearance {
            material Material { }
         }
         geometry Cylinder {
            height 1.0
            radius 0.1
         }
      },
   # Al doilea braţ
      Transform {
         rotation 1.0 0.0 0.0  1.047
         children USE Arm1
      },
   # Al treilea braţ
      Transform {
         rotation 1.0 0.0 0.0  2.094
         children USE Arm1
      }
   ]
}
```

Fişierul `asterix.wrl` construieşte un semn asterix în trei dimensiuni. Obiectul este reprezentat printr-un nod `Group` cu patru fii. Primul nod fiu (`Viewpoint`) poziţionează punctul şi direcţia de observare. Cel de-al doilea nod fiu este un braţ reprezentat printr-un cilindru vertical; cel de-al treilea fiu al nodului `Group` reprezintă al doilea braţ şi se obţine printr-o rotaţie cu 60° (1.047 radiani) a primului braţ relativ la axa x; cel de-al patrulea fiu al nodului `Group` reprezintă al treilia braţ şi se obţine printr-o rotaţie cu 120° (2.094 radiani) a primului braţ relativ la axa x.

Grafurile scenelor corespunzătoare fişierelor `cupola.wrl` şi `asterix.wrl` sunt date în Fig. 8.4.

Nodul `Billboard` defineşte o transformare a nodurilor fii, dar automat adaugă la această transformare o rotaţie astfel încât axa z a sistemului de referinţă al nodurilor fii să fie în permanenţă orientată către punctul de observare. În acest mod, atunci când observatorul se mişcă în scenă, el vede întotdeauna aceeaşi faţă a obiectului. Nodul de tip Billboard se foloseşte pentra redarea copacilor în scenele virtuale. Un copac se reprezintă printr-o singură faţă cu o textură care reproduce imaginea copacului, şi care se roteşte astfel încât observatorul vede faţa texturată şi nu muchiile acesteia.

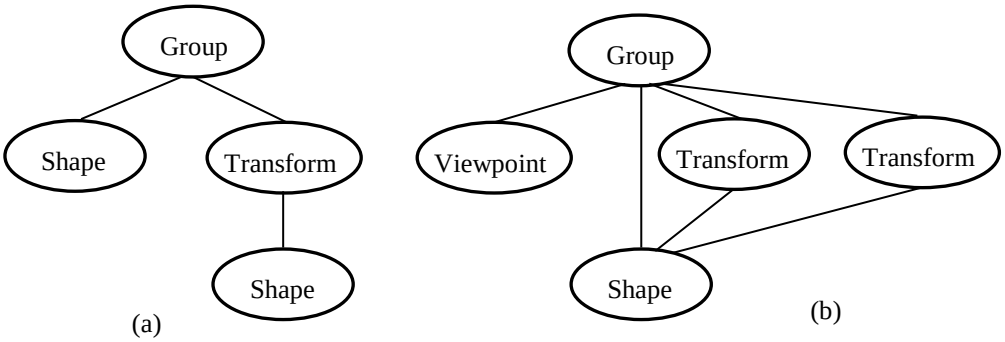


Fig. 8.4 (a) Graful scenei cupola.wrl (b) Graful acenei asterix.wrl.

Nodul `Inline` este un nod de grupare care permite introducerea unuia sau mai multor noduri fii dintr-un alt fişier VRML extern specificat prin codul `URL`. În acest fel, o scenă virtuală se poate compune din mai multe module (fişiere) existente.

8.2.3 Animaţia în VRML

Una din cele mai importante caracteristici ale limbajului VRML este abilitatea de a descrie animaţia obiectelor în scena virtuală. Animaţia VRML poate modifica poziţia, orientarea, dimensiunea obiectelor; culoarea, materialul, textura sau parametrii de amplasare a texturii pe feţele obiectelor; culoarea, poziţia sau direcţia luminilor; şi încă multe alte caracteristici ale scenei virtuale.

Bibliografie

[Aur91] **F. Aurenhammer**, "*Voronoi Diagrams- A Survei of Fundamental Geometric Data Structure*", ACM Computing Survey, Vol. 23 Nr. 3, Sept. 1991.

[Azu97] **R.T Azuma**, "*A survey of Augnented Reality*", Presence: Teleoperators and Virtual Environments, Vol. 6, Nr. 4, pp. 355-385, August 1997.

[Baciu99] **Rodica Baciu** şi **D. Volovici**, *Sisteme de prelucrare grafică*, Editura Albastră, Cluj-Napoca, 1999.

[Burd92] **G. Burdea, J. Zhuang, E. Roskos, D. Silver** şi **N. Langrana**, "*A Portable Dextrous Master with Force Feedback*", Presence- Teleoperators and Virtual Environments, Vol. 1, Nr. 1, pp 18/27, March 1992.

[Burd93] **G. Burdea**, "*Virtual Reality Systems and Applications*", Electro'93 International Conference, Short Course, Edison, NJ, April 1993.

[Burd94] **G. Burdea** şi **P. Coiffet**, *La Réalité Virtuelle*, Edition Hermes, Paris, 1994.

[Burd97] **G. Burdea**, *Force and Touch Feedback for Virtual Reality*, John Willey & Sons, Inc., New York, 1997.

[Dog88] **D. Dogaru**, *Elemente de grafică 3D*, Ed. Ştiinţifică şi Enciclopedică, Bucureşti, 1988.

[Fang95] **Tsung-Pao Fang** şi **Les A. Piegl**, "*Delaunay Triangulation in Three Dimensions*", IEEE Computer Graphics and Applications, Vol. 15, Nr. 5, September 1995, pp. 62-69.

[Fjall93] **Per-Olof Fjallstrom**, "*Evaluation of a Delaunay-based method for surface approximation*", Computer-Aided Design, Vol. 25, Nr. 11, November 1993, pp. 711-719.

[Foley90] **J.D. Foley** şi **A. Van Dam**, *Computer Graphics: Principles and Practices*, 2nd Edn, Addison-Wesley, Reading, Massachusetts, 1990.

[Giga93] **M.A. Gigante**, "*Virtual Reality: Definitions, History and Applications*", Virtual Reality Systems, Academic Press, 1993.

[Gour71] **H. Gouraud,** *Continous Shading of Curved Surfaces*, IEE Trans. on Computers, C-20(6), June 1971, pp. 623-629.

[Ion99] **Felicia Ionescu**, *Principiile Calculului Paralel*, Editura Tehnică, Bucureşti 1999.

[Man95] **C. Manetta şi R. Blade**, "*Glossary of Virtual Reality Terminology*", International Journal of Virtual Reality, Vol.1, Nr. 2, 1995.

[Mold96] **Florica Moldoveanu, Zoea Racoviţă, Ş. Petrescu, G. Hera** şi **M. Zaharia**, *Grafica pe Calculator*, Editura Teora, 1996.

[New81] **W.M. Newman** şi **R.F. Sproull**, *Principles of Interactive Computer Graphics*, McGraw-Hill, New Zork, 1981.

[Prep85] **F.P. Preparata** şi **M.I. Shamos**, *Computational Geometry: an Introduction*, Springer-Verlag, New-York, 1985.

[Rour93] **Joseph O'Rourke**, *Computational Geometry in C*, Cambridge University Press, Boston, Massachusetts, 1993.

[Sher92] **B. Sherman şi P Judkins**, *Glimpses of Heaven, Vision of Hell: Virtual Reality and its implications*, Hodder and Stoughton, London, 1992.

[Suth74] **I.E. Sutherland** şi **G.W Hodgman,** *Reentrant Polygon Clipping*, CACM, January 1974.

[Watt95] **A. Watt**, *3D Computer Graphics*, Addison-Wesley, Reading, Massachusetts, 1995.

[Woo97] **M. Woo, Jackie Neider** şi **T. Davis**, *OpenGL Programming Guide*, OpenGL Architecture Review Board, Addison-Weslez Developres Press, Reading, Massachusetts, 1997.

[Ames97] **A.L. Ames, D.R. Nadeau,** şi **J.L. Moreland**, *The VRML 2.0 Sourcebook*, John Wiley & Sons Inc., New York, 1997.